CLIMATE SCIENCE MODELLING LANGUAGE VERSION 2

USER'S MANUAL

03 Feb, 2007

TABLE OF CONTENTS

Table of Contents 2
Document Change log 3
1. Introduction 4
1.1 Standards-based data modelling 4
1.1.1 Domain Reference Model 4
1.1.2 "Features" 4
1.1.3 Application schema 4
1.1.4 Governance issues 4
1.1.5 Geography Markup Language, GML 5
1.1.6 The 'model-driven approach' 6
1.2 An interoperability model for legacy data 6
1.3 Feature-type principles for the climate sciences 8
1.3.1 Scientific data types 8
1.3.2 'Soft-typing' on phenomenon 9
1.3.3 Conventional portrayal 9
1.4 The 'Observations and Measurements' pattern 9
2. CSML v2 modelling principles (and changes from CSML v1) 11
2.1 More explicit feature types 11
2.2 Observations and Measurements pattern 11
2.3 Behaviour of feature-types: the 'processing affordance' pattern 12
2.4 Dataset structure 14
3. CSML Dataset 16
3.1 CSML FeatureCollection 16
3.2 CSML StorageDescriptor 17
3.3 CSML Dataset wrapper 18
3.4 Local dictionaries 18
3.4.1 Phenomenon 18
3.4.2 Unit definitions 19
3.4.3 Reference system definitions 19
4. Utility types 21
4.1 TimeValueList, TimePositionListType, timePositionList, TimeSeries 21
4.2 CSML ReferenceableGrid 22
5. CSML Feature Types 25
5.1 ReferenceableGrid subclasses 27
5.2 CSML coverage types 27
5.3 CSML Feature Types 28
5.3.1 PointFeature 29
5.3.2 PointSeriesFeature 30
5.3.3 TrajectoryFeature 31
5.3.4 PointCollectionFeature 32
5.3.5 ProfileFeature 33
5.3.6 ProfileSeriesFeature 34
5.3.7 RaggedProfileSeriesFeature 35

5.3.8 SectionFeature 36
5.3.9 RaggedSectionFeature 37
5.3.10 ScanningRadarFeature 38
5.3.11 GridFeature 39
5.3.12 GridSeriesFeature 40
5.3.13 SwathFeature 41
6. File Integration 42
6.1 CSML StorageDescriptor 42
6.1.1 AbstractArrayDescriptor 43
6.1.2 InlineArray 43
6.1.3 ArrayGenerator 44
6.1.4 AbstractFileExtract 44
6.1.5 AggregatedArray 46
6.2 xlink 46
7. Tooling 51
8. Examples 52
9. Appendix: GML Change request for ReferenceableGrid 54
9.1 ReferenceableGrid 54
9.2 GridPointDescription, GridPointDescriptionType, GridPointDescriptionPropertyType 55
9.3 GridOrdinateDescription, GridOrdinateDescriptionType, GridOrdinateDescriptionPropertyType 56
References 59

DOCUMENT CHANGE LOG

Revision history			
Document version	Contributors	Date	Changes
0.1	Andrew Woolf	2005-01-09	CSML Version One
0.2	Andrew Woolf, Dominic Lowe		CSML Version Two

1. INTRODUCTION

The Climate Science Modelling Language (CSML) provides a standards-based semantic model and encoding for representing a range of conceptual information classes of relevance to climate science. These classes may be leveraged to build intelligent services for data subsetting, aggregation, processing, etc. As well, CSML provide a 'wrapper' mechanism to encapsulate legacy file-based data, exposing them instead through the conceptual view.

The motivation and context to CSML has been given in ([i,ii]). Some key elements are reviewed below.

1.1 Standards-based data modelling

An emerging series of international standards for geospatial data, metadata and services ([iii]) provides a timely basis for meeting the CSML data integration requirements ([ii]). The series (comprising some 40 standards) is being developed by Technical Committee 211 (Geographic Information and Geomatics) of the International Organisation for Standardisation (ISO)[1].

1.1.1 Domain Reference Model

The overall scope of these standards is outlined in ISO 19101 ([iv]), which introduces the "Domain Reference Model" – an abstract information architecture for geospatial data infrastructures. At the core of the model is a geospatial "Dataset". A Dataset contains "Feature" instances (see section 1.1.2 below) and related objects, and is described by "Metadata". "Geographic information services" operate on a Dataset, while the logical structure and semantic content of a Dataset is described through an "Application schema" (see section 1.1.3 below).

1.1.2 "Features"

A cornerstone of CSML is to use the "feature" model as its key to integration ([v]).

Defined broadly as an "abstraction of a real world phenomena", a feature may represent any important aspect of a universe of discourse; and may be characterised in terms of its attributes, associations with other features, and operations that may be performed (the so-called "General Feature Model", [vi]). In essence, features provide an object view of data, and may occur as both types and instances. Feature type definitions may be stored for re-use in catalogues (ISO 19110, [vii]). Since features encapsulate important data semantics within communities of practice, such Feature Type Catalogues may be regarded as 'semantics repositories' within an overall information architecture. Feature instances are the primary constituents of geospatial Datasets.

1.1.3 Application schema

While feature types define individual information classes, a "Dataset" is described with an "Application schema" ([iv,vi]). It defines the logical structure and semantic content of a dataset, and specifies the allowable feature types that may be contained. CSML is an application schema of the Geography Markup Language (GML, [viii], see section 1.1.5 below).

1.1.4 Governance issues

The integration framework outlined above aims to capture semantics of important community information types. To be successful, any attempt to apply this framework must

[1] http://www.isotc211.org

engage with the community concerned. Various points of agreement must be established: what are (1) the information objects that should be modelled as features? (2) the precise definition of feature types (i.e. their attributes, associations, and behaviours) (3) common dictionaries (e.g. for physical units, coordinate references systems, physical "phenomena" definitions, etc.) and (4) maintenance procedures for agreed definitions.

Governance also is an important control for typing granularity of features ([ix]). A-priori, it is not always obvious to a designer of feature types how strongly typed they should be (see Figure 1). In general, the more specialised the feature types, the greater will be the number required to capture the spectrum of information types used by the community.

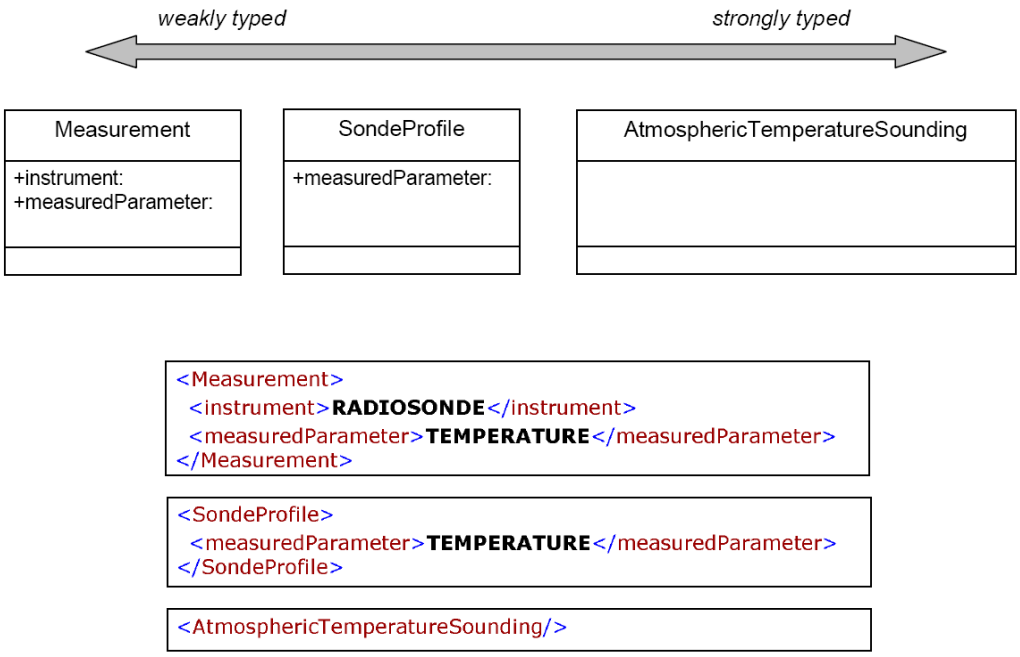


Figure 1: Typing granularity (from [ii])

Within the climate science community, bodies that might be expected to hold a mandate for maintaining detailed Feature Type Catalogues include the UN agencies World Meteorological Organisation (WMO) and International Oceanographic Commission (IOC).

1.1.5 Geography Markup Language, GML

GML ([viii]) provides reference XML schemas for a range of conceptual 'building-blocks' from other ISO standards, e.g. for geometry and topology ([x]), spatial ([xi]) and temporal ([xii]) reference systems, 'coverage' and gridded data ([xiii]), etc.

While GML is a large specification, its purpose is often misunderstood. GML is "an XML grammar written in XML Schema for the description of application schemas as well as the transport and storage of geographic information" ([viii]). The specification notes that "(f)ew applications will require the full range of capabilities described by the GML Schema". The intention with GML is that its schema constructs should be used as building blocks to provide canonical encodings for feature-types designed within a particular application domain. While GML elements *may* be used in arbitrary XML documents, the main purpose is to encode datasets according to an application schema. A valid GML instance document may contain

feature instances or collections, dictionaries (of coordinate reference systems or units of measure etc.), or collections of topological objects.

1.1.6 The 'model-driven approach'

The ISO framework implements a 'model-driven approach' to data encoding. Feature types and datasets are first modelled at a conceptual level using a conceptual schema language (UML, [xiv]). The conceptual UML model then provides an interoperable view of data independent of actual representation. For canonical encoding and exchange, an automated mapping to XML is defined (see [xv], and Annexes E and F of GML [viii]). Indeed, apart from some historical quirks, GML itself is little more than the programmatic XML realisation of UML conceptual models defined in other ISO standards.

CSML provides an additional mechanism for mapping file-based data onto a GML instance, as described next.

1.2 An interoperability model for legacy data

The ambition of CSML to provide standards-based interoperable information models for the climate sciences is in tension with the reality of data management praxis in this domain – very large volume (often tera-scale), file-based data in proprietary or ad-hoc but efficient formats, with a considerable legacy of existing – often elaborate – operational infrastructure built around them. It is infeasible on numerous grounds to imagine warehousing existing data into a new 'interoperable' XML representation.

CSML attempts to strike a middle-ground (Figure 2), and is proposed as a best practice approach for similar problem domains.

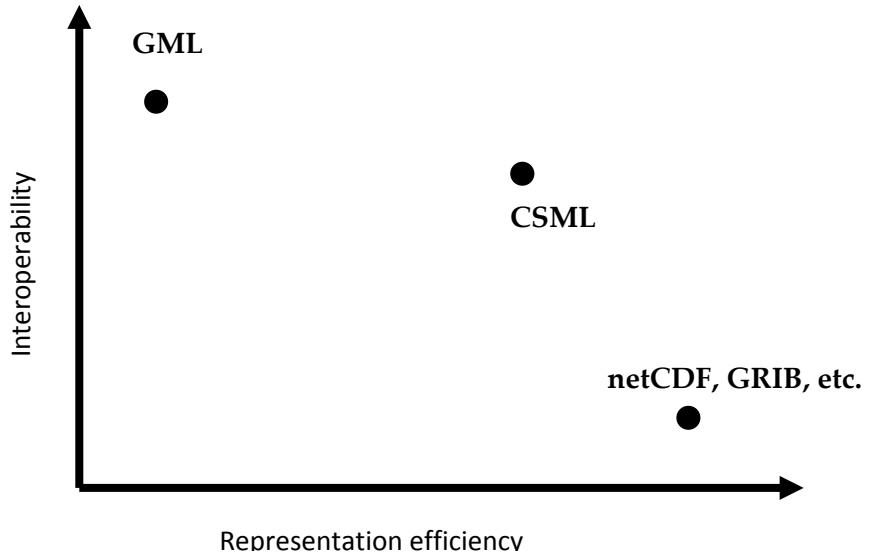


Figure 2: Interoperability *vs* efficiency in the climate sciences

The goal of interoperability suggests the standards-based approach described above – conceptual modelling of important community feature types, with model-driven serialisation into standardised XML representations. On the other hand, compatibility with existing

infrastructure, and efficiency of storage demands the continuing use of community data formats.

In fact, both are needed. The CSML solution formalises the working habits of practitioners, who don't plan their activities in terms of low-level file formats, but rather at a higher conceptual level. For instance a researcher may wish to "calculate the annual average surface temperature over continental Europe". The conceptual model held by the researcher is of a field of temperature data extending over the Earth's surface, and vertically, and varying in time over some number of years. Regardless of the actual representation of the data (be it in one or more netCDF files, or GRIB 2-d layers, or in pages of an Excel spreadsheet), it is clear that this representation can always be *logically mapped onto the conceptual view* (otherwise the researcher's task would not be possible). The mapping might not be straightforward (e.g. it might[2] involve extracting and aggregating arrays of data from many hundreds of files), but it is logically possible.

Thus we arrive at the CSML model for interoperability. It consist of two components (Figure 3):

1. a conceptual model of information capturing essential feature types independent of storage details – an interoperable 'skeleton'; with

2. a mechanism for mapping stored (file-based) data into feature instances to provide the content – adding digital 'flesh' to the conceptual bones.

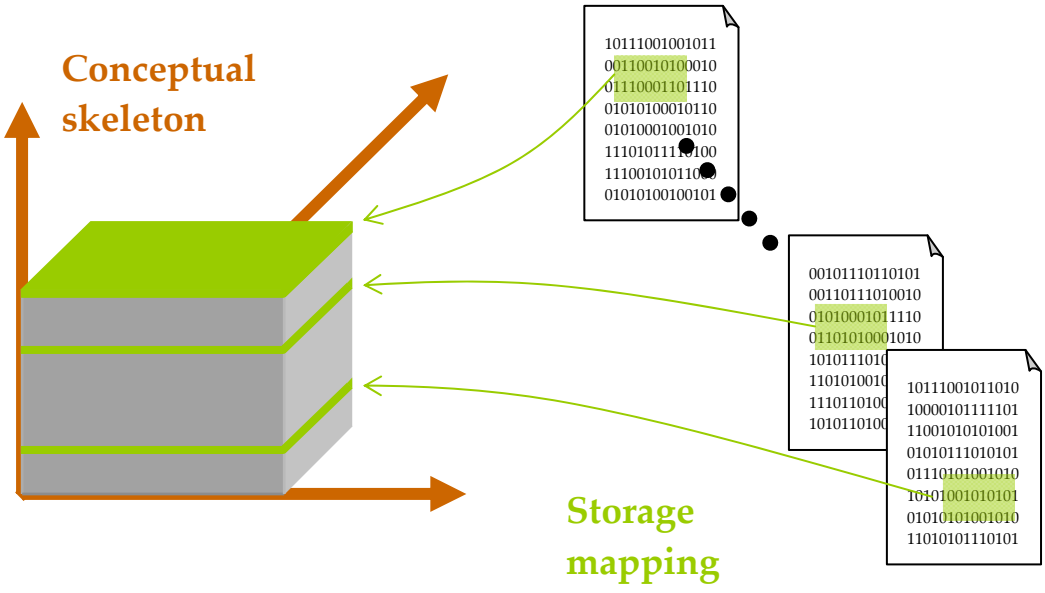


Figure 3: The CSML interoperability model for legacy data

This two-component model for interoperability is represented in CSML through two schemas – a GML application schema for feature instances, and a 'storage descriptor' schema for describing the logical extraction and aggregation of data from storage, with the mapping between them being achieved through the use of xlink (described in section 6.2).

[2] ...and usually does in this example!

1.3 Feature-type principles for the climate sciences

We examine here some of the guiding principles behind the choice of feature-types in CSML.

1.3.1 Scientific data types

Physical processes occur in the natural world across a wide spectrum of spatial and temporal scales, and considerable science informs the design of experimental sampling strategies. It should be no surprise, conversely, that the geometry and topology of observation sets are a fundamental determinant of the scientific uses to which they may be put. Moreover, the properties of the instruments used to generate data themselves place constraints on their interpretation (e.g. as regards accuracy, precision, calibration, required post-processing, etc.). These two factors – the scientific utility of a sampling regime, and the limitations of an observing process – lead to a natural, scientifically important, classification of data types along these axes. Quite often the two are highly correlated (certain instruments generate certain samplings), and so scientific communities of practice adopt more abstract conceptual information classes that nevertheless reflect artefacts of sampling or instrument-type. This is particularly evident in the climate sciences, where broad information classes based on measurement-set geometry and topology have almost universal acceptance. The following examples are illustrative.

The US National Oceanographic and Atmospheric Administration (NOAA) is developing a plan for a Global Earth Observing Integrated Data Environment (GEO-IDE) to integrate measurements, data and products and create interoperability across data management systems. In the GEO-IDE Concept of Operations (https://www.nosc.noaa.gov/dmc/docs/NOAA_GEO-IDE_CONOPS-v3-3.pdf), the following 'structural data types' are defined: *Grids*, *Moving-sensor multidimensional fields*, *Time series*, *Profiles*, *Trajectories*, *Geospatial Framework Data*, *Point Data*, *Metadata*.

The ESRI 'ArcMarine' Data Model for marine data includes classes like *Instant*, *Location Series*, *Time Series*, *Profile Line*, *Track*, *Sounding*, *Survey*, *{Ir}Regularly Interpolated Surfaces*, *Mesh Volume*, etc.

File formats such as netCDF and NASA Ames utilise data models that reflect these structures (e.g. netCDF four-dimensional gridded lat-lon-height-time variables, or NASA Ames time-series at a point).

Based on this observation, CSML feature types are classified primarily around geometric and topologic structure, and not the semantics of the observable or measurand. This is consistent with the governance principle of Atkinson. It would, for example, be outside the scope of CSML to define highly specialised feature types for universal exchange of met observations (SYNOPs, METARs etc.). The definition of such specialised feature types is rightly the domain of international organisations such as the WMO. It is expected, however, that transformations would be possible from such specialised feature types to those of more general ambition defined here.

The CSML feature types do not carry any explicit topologic descriptions at present. However GML provides rich schemas for describing topology – a requirement may arise in the future to add this information to CSML.

1.3.2 'Soft-typing' on phenomenon

If two features are structurally identical, except for the physical 'phenomenon' of interest (temperature, salinity, wind vector, humidity, etc) then they are modelled in CSML as the same feature type. For example, both a vertical wind profile and an atmospheric temperature sounding have similar characteristics (in terms of attributes and data handling operations that may be performed), differing primarily in the distinguishing physical parameter (vector wind *vs* temperature). Their representations are collapsed, in CSML, into the same feature type.

This principle ('soft-typing' on phenomenon) has been identified as appropriate for GML modelling of coverage-type data in operational meteorology ([xvi]).

1.3.3 Conventional portrayal

Most climate science data has a conventional portrayal used by practitioners (e.g. model output is typically displayed as shaded 2-d slices, an atmospheric sounding as a line graph against height in the vertical, a marine temperature section as a 2-d contoured field against depth and ship track distance, etc.). A workable minimum granularity for CSML feature types is determined by applying a discriminant of "sensible plotting": there should be sufficient detail within a feature type – and sufficient difference between feature types – to enable in-principle unsupervised rendering, in the conventional manner. This criterion is somewhat loose, and it remains to be seen in practice the extent to which it is satisfied with the feature types chosen. In that sense, the principle may play a more important role in evaluation than in design.

1.4 The 'Observations and Measurements' pattern

The Open Geospatial Consortium's Observations and Measurements best practice paper ([xvii]) provides a conceptual model and encoding for observations and measurements.

Under this model, an *observation* is an event whose *result* is an estimate of the value of some *property* of a *feature-of-interest* obtained using a specified *procedure* (Figure 4). Each of the major classes in the model may be specialised for a particular application domain.

The observed property of an observation may be modelled using the O&M Phenomenon taxonomy (Figure 5).

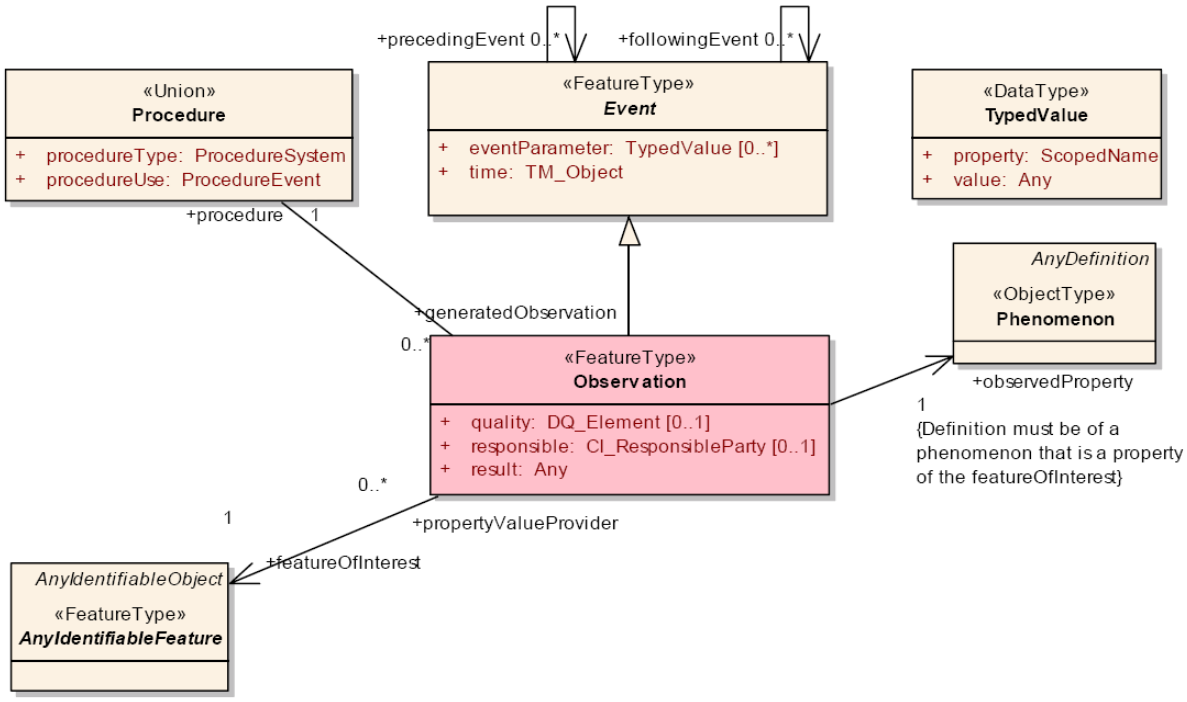


Figure 4: The Observations and Measurements model (from [xvii])

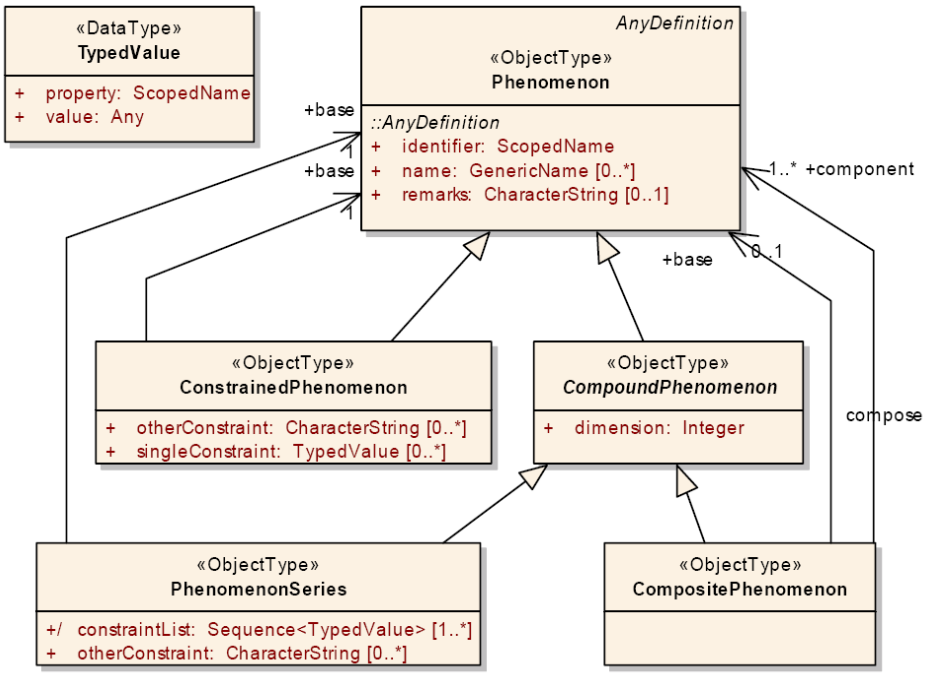


Figure 5: The O&M Phenomenon model (from [xvii])

2. CSML V2 MODELLING PRINCIPLES (AND CHANGES FROM CSML V1)

Version Two of CSML includes a number of significant improvements over Version One ([xviii, xix]). These are outlined here.

2.1 More explicit feature types

CSML v1 used a 'composite-domain pattern' to separate the feature-type geometry into a 'reference' component and a 'complement'. For example, a ship's cruise track provides a reference geometry from which CTD measurements are taken along a complementary vertical geometry down through the water column. In addition, the 'reference' geometry was overloaded in CSML v1, with some sophisticated heuristics required to infer the information class. For instance a 'ProfileSeries' feature represented a series of profile measurements away from some reference geometry. The reference geometry could be a single point with a series of angular 'look directions' (for a scanning radar), or a series of time instants (for a vertical sounding radar time-series), or a full trajectory in time and space (for a marine section along a ship's cruise track).

In CSML v2, the six core feature types of CSML v1 have been refactored into a series of 13 more explicit feature types, as per Table 1. In addition, a Swath feature has been added.

Table 1: CSML feature type changes, v1 to v2

CSML v1 feature type	CSML v2 feature type
Point	Point
PointSeries	PointSeries
	Trajectory
	PointCollection
Profile	Profile
ProfileSeries	ProfileSeries
	RaggedProfileSeries
	Section
	RaggedSection
	ScanningRadar
Grid	Grid
GridSeries	GridSeries
	Swath

2.2 Observations and Measurements pattern

CSML v1 used only the 'Phenomenon' class of the O&M model. CSML v2, however, is re-factored to be fully-compatible with the model and the infrastructure being developed around it, e.g. web services including the Sensor Observation Service, and instrument descriptions using SensorML.

The general O&M pattern was described earlier in section 1.4 and Figure 4. It is applied in CSML by modelling the following three components (Figure 6):

1. a *CSML feature type*; with an associated
2. physical *parameter*; and the
3. *value* of this parameter as a CSML coverage.

Thus, for example, a CSML *Grid Feature* may be measuring the parameter *temperature*, with the temperature values supplied though a *gridded coverage*.

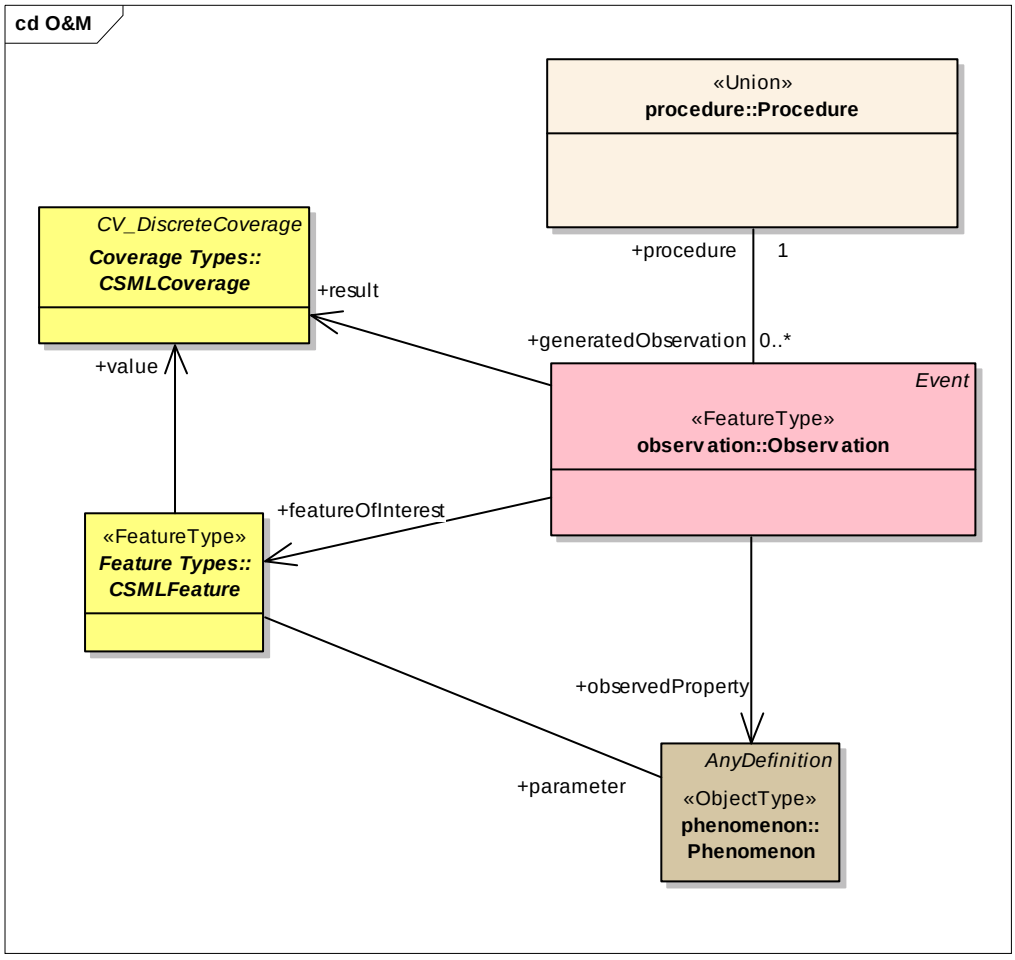


Figure 6: The 'Observations and Measurements' view of CSML

These three components then provide directly the O&M *feature-of-interest*, *observed property*, and *observation result*, respectively. Not explicitly modelled in CSML is the O&M *procedure* used to generate the result.

2.3 Behaviour of feature-types: the 'processing affordance' pattern

A UK MetOffice-sponsored workshop ([xvi]) on GML modelling for operational meteorology recognised that mere GML modelling and representation of data types omitted an important element of the General Feature Model – the operations that may be performed on an object. For instance, a four-dimensional timeseries of gridded NWP data supports an operation to extract from it a pseudo radiosonde observation. This is easy enough to model conceptually

(e.g. through an operation extractRadiosonde() on the UML feature class), but is not represented in the GML serialisation.

The notion of 'processing affordance' was proposed by the workshop as a means to capture the joint concept of the operations allowed on a feature-type together with the structural elements required to support them (e.g. extracting a pseudo-radiosonde is a feature-type operation that requires a four-dimensional gridded timeseries structure to support it). Thus, processing affordance may be regarded as a binding between the operations that may be performed on an object with structural properties of the object required to support those operations.

This matches closely the conventional UML concept of a «type»:

> "A Type is used to specify a domain of objects together with operations applicable to the objects without defining the physical implementation of those objects. A Type may not include any methods, but it may provide behavioural specifications for its operations. It may also have attributes and associations that are defined solely for the purpose of specifying the behaviour of the type's operations." ([xx])

The match is not exact, since the attributes and associations of a «type» are not required to be inherited by an implementation class.

It was recognised at the workshop that CSML might benefit from modelling the operations supported by the sampling geometries that characterise CSML feature classes. This has been implemented in CSML v2 using the conventional UML stereotype «type» (avoiding any requirement for more radical extensions to the general modelling paradigm). Figure 7 shows the CSML modelling of processing affordance through a UML «type» realised by a feature.

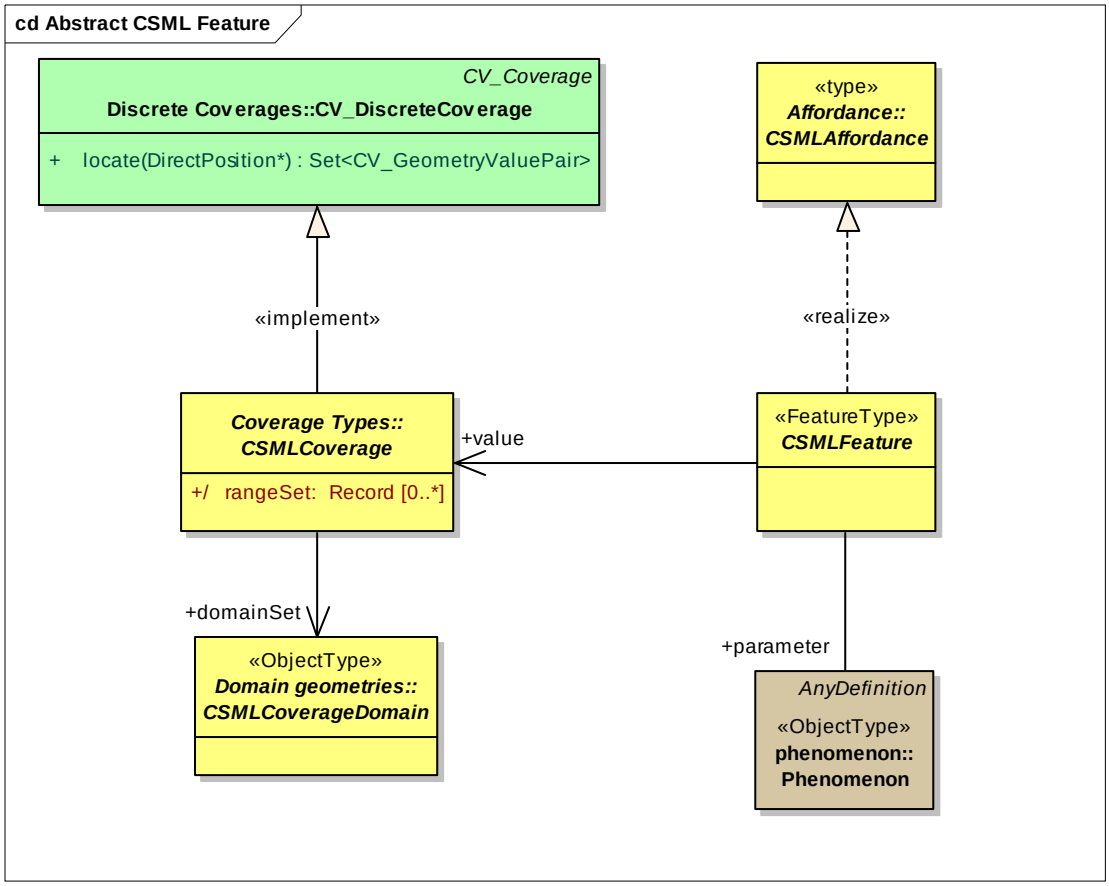


Figure 7: Implementation of 'processing affordance' in CSML through UML «type»

Two other modelling points are worth noting from Figure 7: CSML defines specialised domain geometries for its respective coverages, and the CSML coverage classes are logical implementations of the ISO 19123 CV_DiscreteCoverage coverage class hierarchy.

2.4 Dataset structure

As mentioned above (section 1.2), CSML marries a conceptual 'features view' of information with a mechanism for mapping onto this view legacy file-based data. In CSML v1, both components were part of the one schema and included together in a CSML (i.e. GML instance) document. However, the 'storage descriptor' component cannot validly extend any semantic construct provided within GML – there is nothing in GML concerned with data storage artefacts[3].

Thus, in CSML v2, the CSML feature types and GML application schema are separated from the storage descriptor schema. An instance of the CSML application schema is a valid GML document that may *refer* (using an xlink mechanism) to CSML storage descriptors in a logically separate document.

[3] With the sole exception of a very limited and inadequate representation of coverage range data – which, if encoded within a single record-interleaved (but not band-sequential, etc.) file, may be referenced through the gml:File element.

For purposes of maintenance and convenience, a simple wrapping schema provides a root 'Dataset' element that includes both a GML document and a storage descriptor document (Figure 8). These various elements are described in more detail in the next chapter.

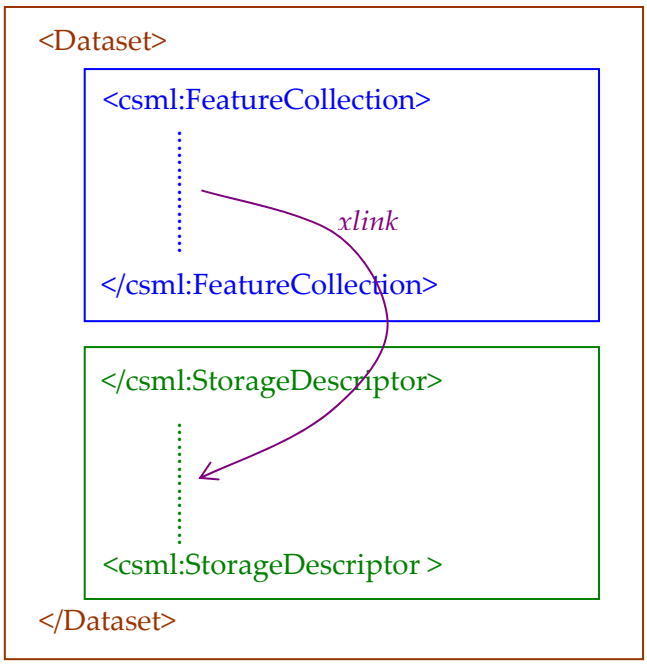


Figure 8: CSML v2 Dataset structure

3. CSML DATASET

A brief overview of the CSML Dataset structure was given above (2.4). It is described here in more detail. A CSML Dataset consists of two components – a CSML FeatureCollection, and a CSML StorageDescriptor.

3.1 CSML FeatureCollection

A CSML FeatureCollection is modelled as shown in Figure 9. As required by GML ([viii, §9.1.10, §2.3]), a feature collection is modelled itself as a feature type.

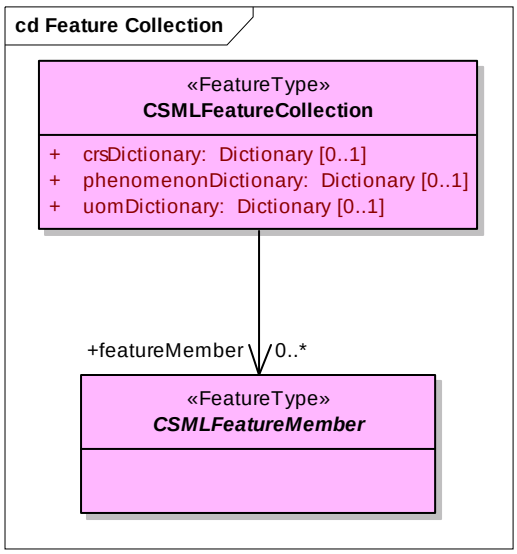


Figure 9: CSML FeatureCollection

The FeatureCollection contains optional local definitions of units, reference systems, and phenomena, as well as a collection of feature instances. Since a FeatureCollection also derives from the gml:AbstractFeatureType, it inherits the following standard object attributes:

- **id**: a handle for the feature collection within the CSML document
- **metaDataProperty**: may be used to link to a metadata record describing the feature collection.
- **description**: a text description of the feature collection.
- **identifier**: an identifier – unique within the application domain – to be used in references to the feature collection, together with a mandatory authority code
- **name**: a descriptive label for the feature collection (with optional authority)
- **boundedBy**: a bounding envelope for the spatiotemporal extent of the feature collection

An example skeleton XML encoding is outlined in Figure 10.

```
<CSMLFeatureCollection gml:id="FeatureCollection">
    <gml:description>Test CSML feature collection.</gml:description>
    <gml:identifier codeSpace="http://ndg.nerc.ac.uk">FC000000001</gml:identifier>
```

```
    <gml:name>NDG Feature Collection</gml:name>
    <gml:boundedBy>
        <gml:EnvelopeWithTimePeriod srsName="urn:EPSG:geographicCRS:4979">
            <gml:lowerCorner>-10 15 0</gml:lowerCorner>
            <gml:upperCorner>30 65 5000</gml:upperCorner>
            <gml:timePosition>1998-01-01</gml:timePosition>
            <gml:timePosition>2003-12-31</gml:timePosition>
        </gml:EnvelopeWithTimePeriod>
    </gml:boundedBy>

    <!--=== Local reference system definitions ===-->
    <crsDictionary>
        .....
    </crsDictionary>

    <!--=== Local phenomenon definitions ===-->
    <phenomenonDictionary>
        .....
    </phenomenonDictionary>

    <!--=== Local physical unit definitions ===-->
    <uomDictionary>
        .....
    </uomDictionary>

    <!--=== Collection of feature instances ===-->
    <featureMember>
        <!-- feature instance -->
    </featureMember>

    <featureMember>
        <!-- another feature instance -->
    </featureMember>

</CSMLFeatureCollection>
```

Figure 10: Example skeleton XML encoding of CSML feature collection

The local dictionaries for reference systems, phenomena and units of measure are each structured as GML dictionaries, as detailed in section 3.4.

The collection of feature instances follows the standard GML mechanism ([viii, §9.1.10]) by defining the featureMember association to an AbstractFeatureMember.

3.2 CSML StorageDescriptor

GML provides two mechanisms for defining property content – either inline or 'by-reference' via xlink (see [viii, §7.2.3]). CSML exploits the 'by-reference' device (together with a speculative pattern on the use of xlink) to assert that property content is derived from legacy storage. However, there is not generally a direct mapping between storage data models (e.g. netCDF or GRIB files, or relational tables) and the conceptual feature types of CSML; thus an additional model is needed for defining blocks of content parametrically derived from one or more storage elements. This is the CSML StorageDescriptor schema, instantiated in an XML document logically separate from the CSML FeatureCollection, but closely related to it through xlink references between the two (Figure 8).

The details of the StorageDescriptor are described in section 6.1 while the use of xlink is outlined in section 6.2. Figure 11 shows a skeleton XML encoding for the CSML StorageDescriptor.

```
        <CSMLStorageDescriptor>
            <descriptor>
            .....
```

```
    </descriptor>
    <descriptor>
    .............
    </descriptor>
</CSMLStorageDescriptor>
```

Figure 11: Skeleton XML encoding for CSML StorageDescriptor

3.3 CSML Dataset wrapper

The CSML FeatureCollection is a valid GML collection of features and can form the root of valid GML instance document. On the other had, the CSML StorageDescriptor does not inherit from any GML construct, and may not form any part of a valid GML instance. Logically, the CSML FeatureCollection and StorageDescriptor are root elements of two separate XML documents.

However, the two documents are in fact closely related – the StorageDescriptor describes the construction from stored data of logical feature instances in the FeatureCollection. From a maintenance perspective it is essential that they be tightly coupled.

A simple convenience schema is therefore defined which wraps a FeatureCollection and StorageDescriptor together into a CSML Dataset. The structure was shown earlier in Figure 8.

3.4 Local dictionaries

As described in section 3.1 above, a CSML FeatureCollection may include local definitions of reference systems, phenomena, and units of measure using the gml:Dictionary structure ([viii, §16], Figure 12) as the child of csml:crsDictionary, csml:phenomenonDictionary or csml:uomDictionary elements respectively. Each of the three types of dictionary are described below.

The following standard properties are available for a GML Dictionary: id, description, identifier, name.

```
<gml:Dictionary gml:id="CSML Local CRS Dictionary">
    <gml:identifier codeSpace="http://ndg.nerc.ac.uk">CRS0000001</gml:identifier>
    <gml:dictionaryEntry>
        .................
    </gml:dictionaryEntry>

    <gml:dictionaryEntry>
        .................
    </gml:dictionaryEntry>
</gml:Dictionary>
```

Figure 12: XML structure of GML dictionary

3.4.1 Phenomenon

CSML feature types are representations of a physical phenomenon (temperature, wind, salinity, etc.). The definition of physical phenomena is referred typically to external authorities (for instance, see section XXX for an example GML encoding of the 'CF standard name table' widely used in atmospheric sciences). However, local definitions may be prescribed, based on the O&M conceptual model of Figure 5.

3.4.1.1 Phenomenon

A phenomenon definition is characterised by a local id, optional description, namespace-qualified identifier and optional name. An example XML encoding is shown in Figure 13 (this may provide the immediate content of a gml:dictionaryEntry).

```
<swe:Phenomenon gml:id="rainfall">
    <gml:description>Liquid precipitation measured with raingauge.</gml:description>
    <gml:identifier codeSpace="http://ndg.nerc.ac.uk">gauge_rainfall</gml:identifier>
    <gml:name>rainfall</gml:name>
</swe:Phenomenon>
```

Figure 13: XML encoding of Phenomenon

3.4.1.2 CompositePhenomenon

A composite phenomenon may be defined, for example vector winds. Each component is itself a Phenomenon. Example XML encodings may be found in [xvii].

3.4.1.3 ConstrainedPhenomenon

A phenomenon with multiple components derived by applying constraints to some base phenomenon may be characterised as a ConstrainedPhenomenon. For example, multi-spectral radiance bands apply wavelength constraints to a base 'radiance' phenomenon; particle size distributions are classified by bands of particle size range. Example XML encodings may be found in [xvii].

3.4.2 Unit definitions

Normally, units defined by external authorities are used (see section XXX for an example encoding of the 'udunits' system of units often used in the atmospheric sciences). If local units need to be defined, the GML units schema is used ([viii, §17.2]). An example XML encoding is shown in Figure 14, and may provide the content for a gml:dictionaryEntry element.

```
<gml:UnitDefinition gml:id="psu">
    <gml:description>Conventional practical salinity units.</gml:description>
    <gml:identifier codeSpace="http://ndg.nerc.ac.uk/units">practical_salinity_units</gml:identifier>
    <gml:name>practical salinity units</gml:name>
    <gml:quantityType>sea water salinity</gml:quantityType>
    <gml:catalogSymbol codeSpace="http://ndg.nerc.ac.uk/units">psu</gml:catalogSymbol>
</gml:UnitDefinition>
```

Figure 14: XML encoding of unit definition

3.4.3 Reference system definitions

For defining coordinate reference systems, the schemas of GML chapter 13 (spatial, based on [xi]) and 15 (temporal, based on [xii]) are used, and should be consulted for details.

Briefly, a coordinate reference system is defined by both a coordinate system and a datum (Figure 15).

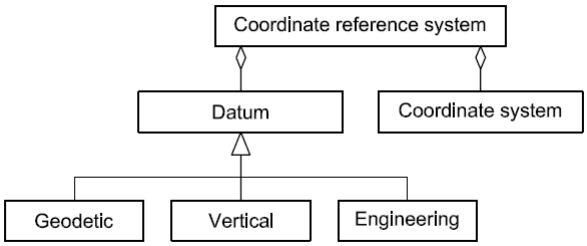


Figure 15: Conceptual model for coordinate reference system (taken from ISO 19111)

A coordinate system is defined by an ordered set of axes, including their directions and units. A datum provides the reference for the axes. A number of coordinate system subclasses are

specified by GML, for instance geographic (horizontal latitude-longitude), projected (for map projections), vertical, and engineering (for local application, e.g. relative to a moving platforms).

Temporal reference systems are subclassed as shown in Figure 16. An 'ordinal reference system' is used, for instance, for specifying time relative to geological eras. A temporal 'coordinate system' refers a time instance to an origin, while calendars and clocks specify an absolute date and time with respect to a defined calendar system.

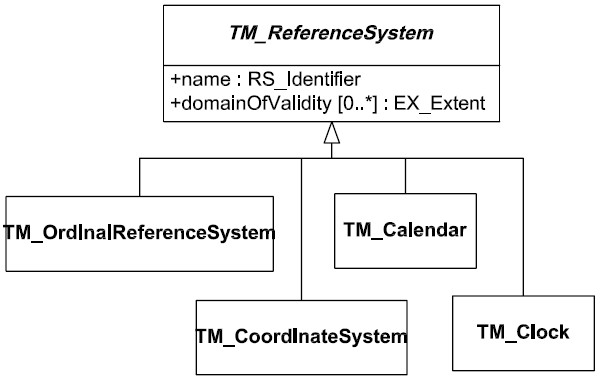


Figure 16: Temporal reference systems (from ISO 19108)

XML encodings for defining reference systems are provided in GML ([viii]), and not reproduced here.

4. UTILITY TYPES

CSML defines some utility types not provided by GML. These are described here.

4.1 TimeValueList, TimePositionListType, timePositionList, TimeSeries

CSML has a requirement to represent efficient lists of time instants. Several constructs are defined for this purpose, mostly by analogy with existing GML structures.

A csml:TimeValueList is defined as a list of gml:TimePositionUnion, Figure 17. This enables a compact representation for a list of time positions.

```
<simpleType name="TimeValueList">
    <annotation>
        <documentation>Provides a list of GML's TimePositionUnion (see GML 3.2, sect 15.2.2.7)</documentation>
    </annotation>
    <list itemType="gml:TimePositionUnion"/>
</simpleType>
```

Figure 17: CSML TimeValueList

By direct analogy with gml:TimePositionType, the CSML TimePositionListType (Figure 18) supplements the TimeValueList with frame and calendar attributes to support multiple time coordinate systems.

```
<complexType name="TimePositionListType">
    <annotation>
        <documentation>By direct analogy with gml:TimePositionType, adds attributes to csml:TimeValueList to
support multiple time coordinate systems (see GML 3.2, sect 15.2.2.7).</documentation>
    </annotation>
    <simpleContent>
        <extension base="csml:TimeValueList">
            <attribute name="frame" type="anyURI" default="#ISO-8601"/>
            <attribute name="calendarEraName" type="string"/>
            <attribute name="indeterminatePosition" type="gml:TimeIndeterminateValueType"/>
        </extension>
    </simpleContent>
</complexType>
```

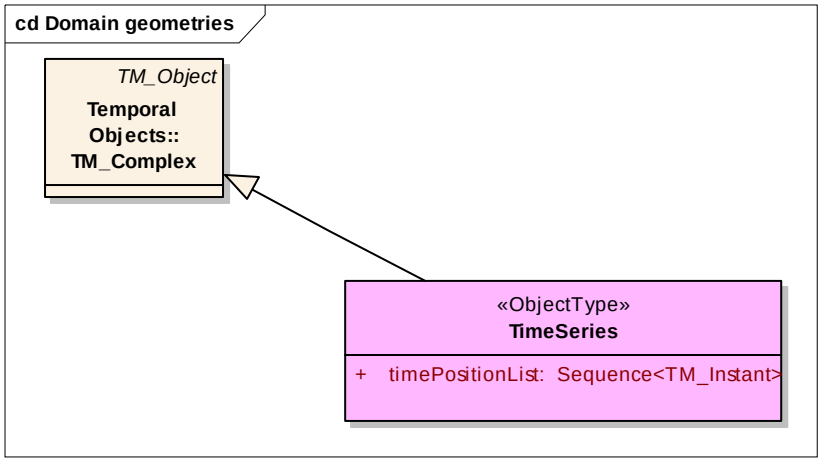
Figure 18: CSML TimePositionListType

Once more, by analogy with gml:timePosition, a CSML element timePositionList is defined (Figure 19).

```
<element name="timePositionList" type="csml:TimePositionListType">
    <annotation>
        <documentation>By direct analogy with gml:timePosition, creates a list version of the same (see GML 3.2,
sect 15.2.2.7).</documentation>
    </annotation>
</element>
```

Figure 19: CSML timePositionList

An application of these definitions is the CSML TimeSeries object (Figure 20).




```
<complexType name="TimeSeriesType">
    <complexContent>
        <extension base="gml:AbstractTimeComplexType">
            <sequence>
                <element ref="csml:timePositionList"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<element name="TimeSeries" type="csml:TimeSeriesType" substitutionGroup="gml:AbstractTimeComplex"/>

<complexType name="TimeSeriesPropertyType">
    <sequence minOccurs="0">
        <element ref="csml:TimeSeries"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
```

Figure 20: CSML TimeSeries

Figure 21 gives an example encoding a time series of 30 fortnightly time instants in a calendar of 30-day months (assumed defined elsewhere).

```
<TimeSeries gml:id="times0001">
    <timePositionList frame="ndg:calendar:360day">
2007-01-15 2007-01-30 2007-02-15 2007-02-30 2007-03-15 2007-03-30 2007-04-15 2007-04-30 2007-05-15 2007-05-30
2007-06-15 2007-06-30 2007-07-15 2007-07-30 2007-08-15 2007-08-30 2007-09-15 2007-09-30 2007-10-15 2007-10-30
2007-11-15 2007-11-30 2007-12-15 2007-12-30 2008-01-15 2008-01-30 2008-02-15 2008-02-30 2008-03-15 2008-03-30
    </timePositionList>
</TimeSeries>
```

Figure 21: CSML TimeSeries example

4.2 CSML ReferenceableGrid

ISO 19123 defines three classes of quadrilateral grid geometry – a non-geo-referenced Grid (CV_Grid), a grid with regular spacing (CV_RectifiedGrid), and a grid with irregular spacing (CV_ReferenceableGrid), Figure 22.

GML, however, provides implementations only of CV_Grid (gml:Grid) and CV_RectifiedGrid (gml:RectifiedGrid). The case of grid geometries with unequal spacing has no GML encoding mechanism. However, this type of geometry is crucial in the climate sciences; the following is a non-exhaustive indicative list of examples:

- weather models with an enhanced resolution over some region of interest

- ocean models with displaced poles to avoid the meridional convergence singularity
- boundary-fit models (e.g. for coastal dynamics or river flow)
- non-rectified remote-sensed imagery

CSML provides an implementation of the ISO 19123 CV_ReferenceableGrid, and this has been submitted to OGC as a GML change request in document number 06-160 ([xxi]).

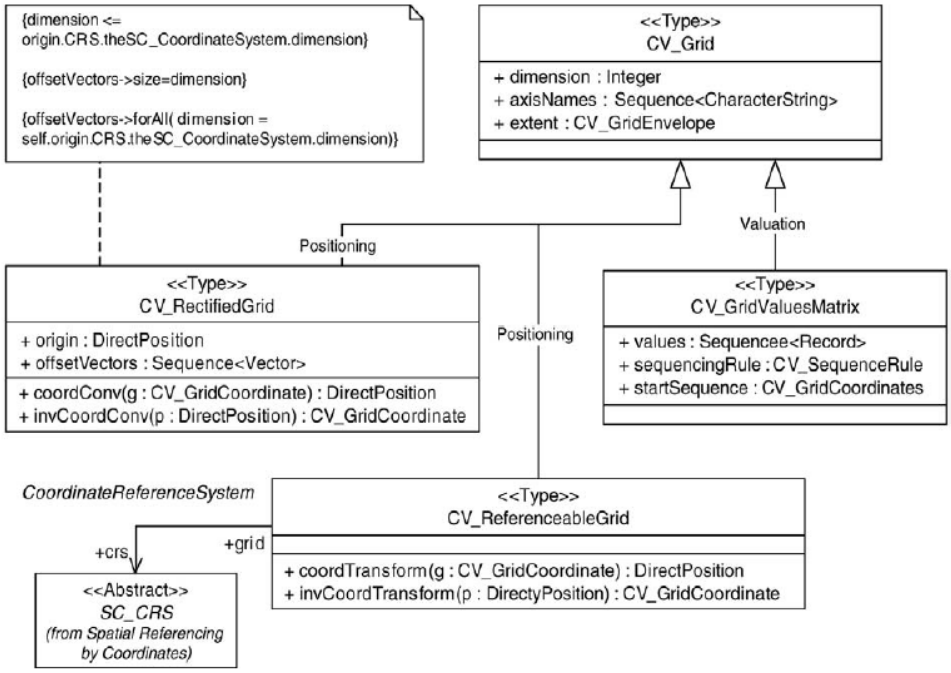


Figure 22: ISO 19123 classes of quadrilateral grid geometry

The text from this change proposal is reproduced in the Annex at section 10 of this manual, but Figure 23 below shows the UML model for CSML's implementation. Instance examples are given in Figure 52 and Figure 53 of the Annex.

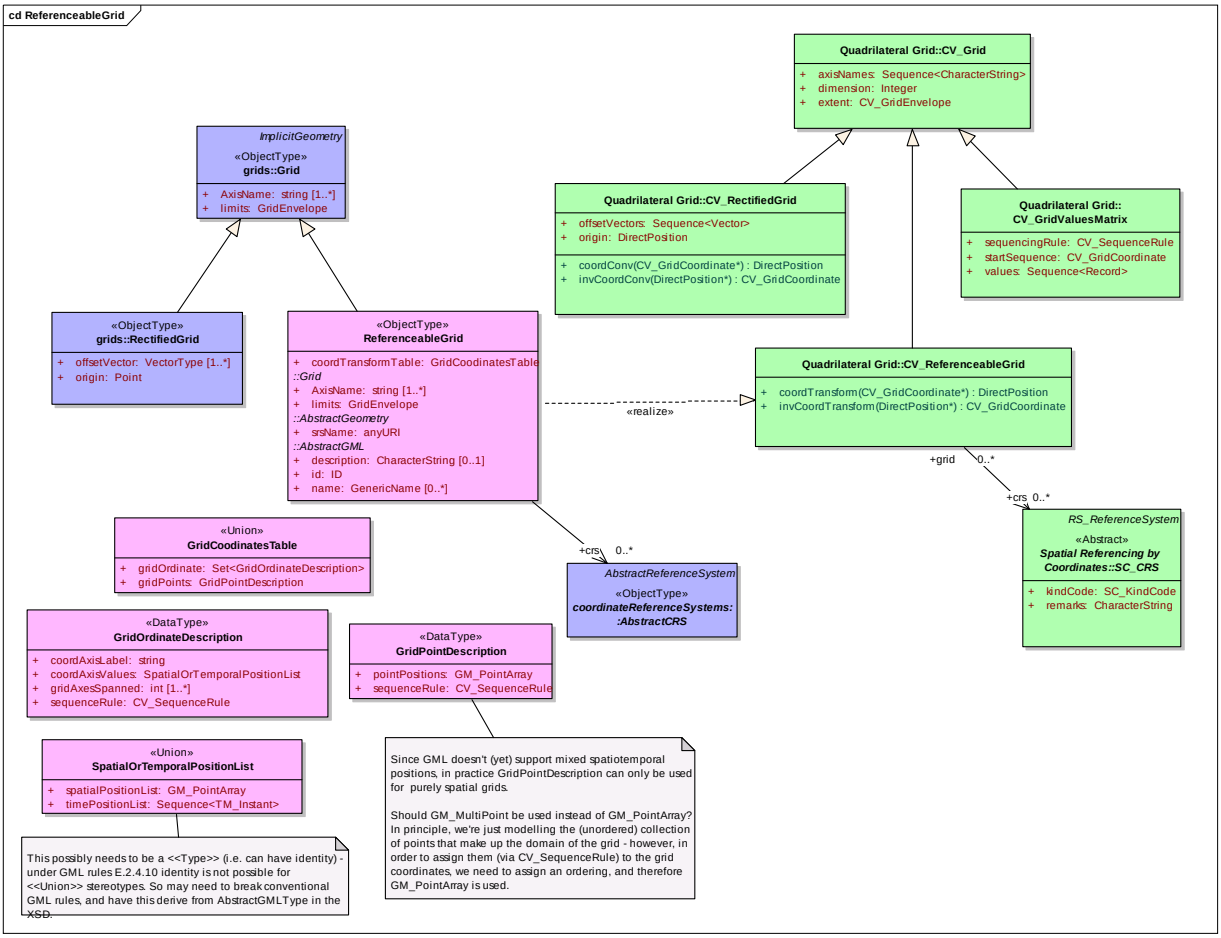
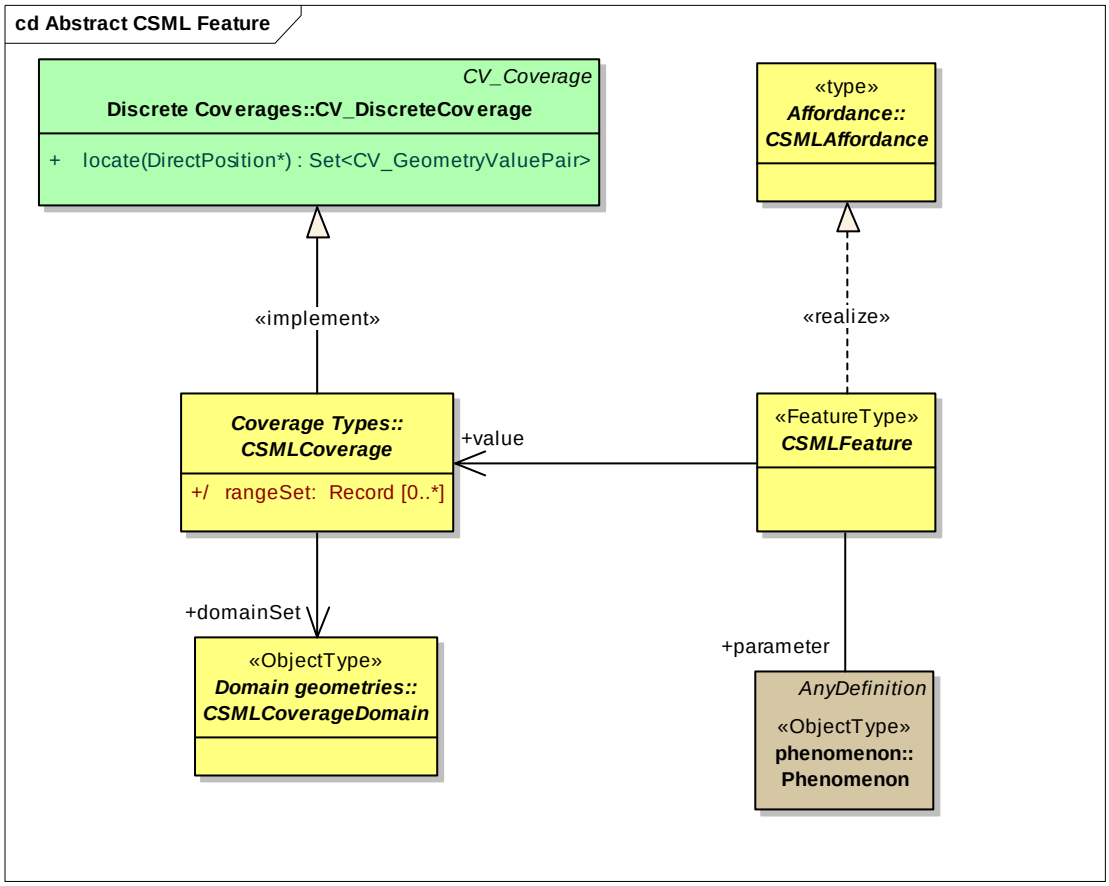


Figure 23: UML model for CSML ReferenceableGrid implementation

5. CSML FEATURE TYPES

The general model for all CSML feature types was illustrated earlier, and is reproduced here for convenience:



Important elements to note are:

- A CSML Feature Type *realises a «type»* (representing a 'processing affordance'). Thus, the operations of the «type» are inherited, and in the CSML application of this pattern attributes of the «type» are copied down (not strictly required by UML)
- A CSML Feature Type represents some observed or simulated *physical parameter* (modelled through the O&M Phenomenon class taxonomy).
- The *value* of this parameter is provided by a CSML 'coverage'. A coverage is, conceptually, a mapping from some spatiotemporal domain to a value range (Figure 24).
- CSML feature types are distinguished primarily on the basis of the *geometry of the coverage domain*. There is a strong relationship between these 'sampling geometries' and the Feature Type operations that are afforded.

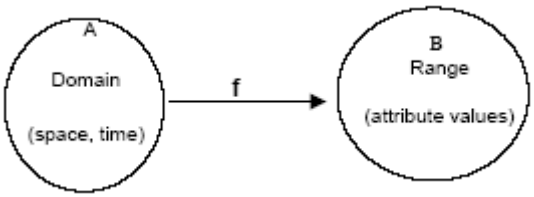


Figure 24: GML coverage model

The set of 13 CSML Feature Types are listed in Table 2 below.

Table 2: CSML Feature Types

Feature type	Description	Example
PointFeature	Single point measurement.	raingauge measurement
PointSeriesFeature	Time-series of single datum measurements at a fixed location in space.	tidegauge, rainfall timeseries
TrajectoryFeature	Measurement along a discrete path in time and space.	surface salinity along a ship's cruise track; atmospheric aerosols along an aircraft's flight path
PointCollectionFeature	Collection of distributed single datum measurements at a particular time	2m temperatures measured at weather stations across the UK at 0600z.
ProfileFeature	Single 'profile' of some parameter along a vertical line in space.	wind sounding, XBT, CTD, radiosonde
ProfileSeriesFeature	Time-series of profiles on fixed vertical levels at a fixed location	vertical radar timeseries, thermistor chain timeseries
RaggedProfileSeriesFeature	Time-series of unequal-length profiles, but on fixed vertical levels, at a fixed location	repeat daily balloon soundings of atmospheric temperature from the same location
SectionFeature	Series of profiles from positions along a trajectory in time and space.	shipborne ADCP
RaggedSectionFeature	Series of profiles of unequal length along a trajectory in time and space	marine CTD measurements along a ship's cruise track
ScanningRadarFeature	Backscatter profiles along a look direction at fixed elevation but rotating in azimuth	weather radar
GridFeature	Single time-snapshot of a gridded field.	gridded analysis field
GridSeriesFeature	Time-series of gridded parameter fields	numerical weather prediction model, ocean general circulation model
SwathFeature	Two-dimensional grid of data along a satellite ground-path	AVHRR satellite imagery

5.1 ReferenceableGrid subclasses

The CSML ReferenceableGrid structure plays an important role in modelling the geometry of the feature types. A number of the CSML coverage domain geometries derive from CSML ReferenceableGrid, restricting on dimensionality of the grid, or coordinate reference system, or both. Figure 25 illustrates the ReferenceableGrid subclasses used in CSML.

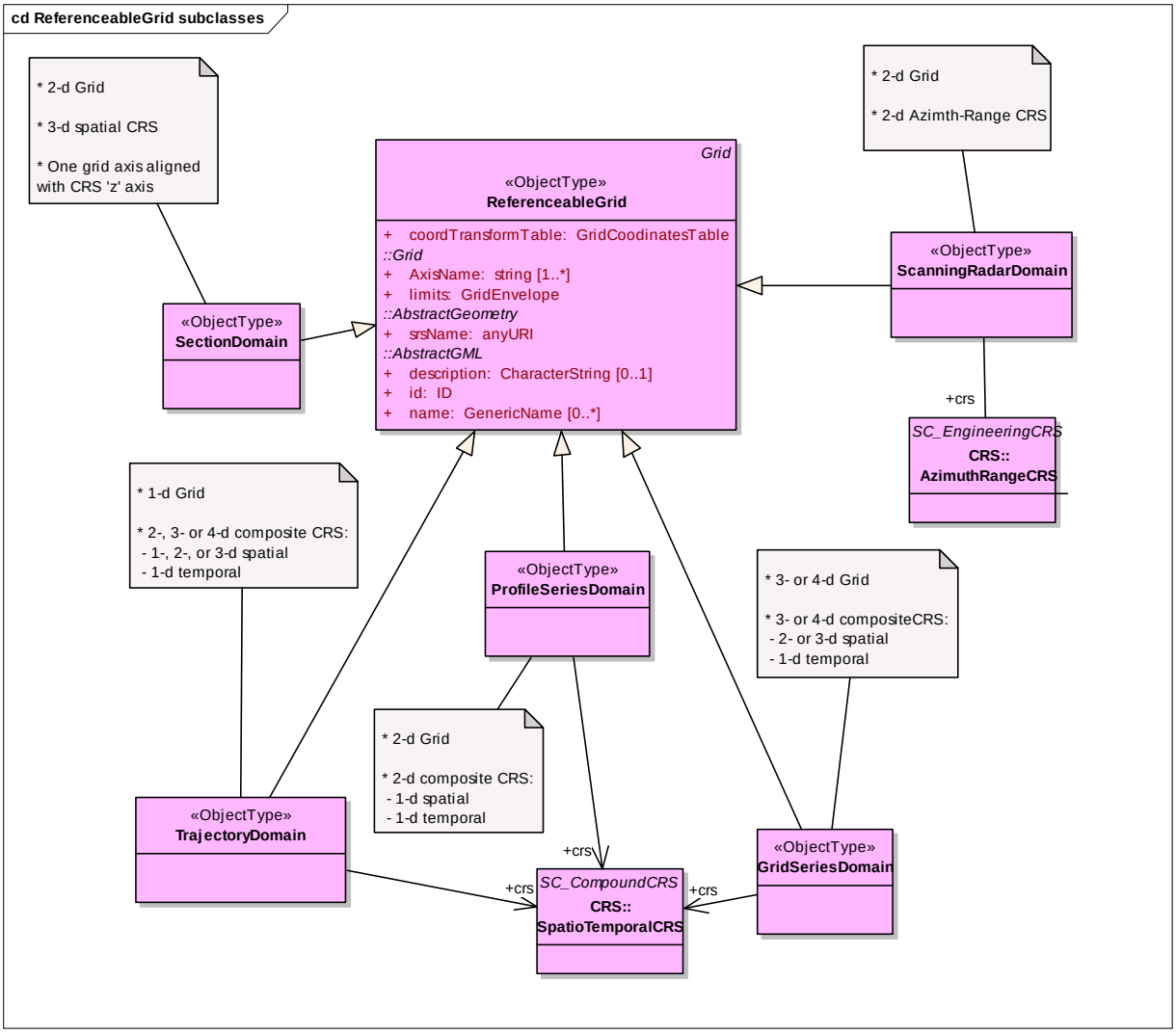


Figure 25: ReferenceableGrid subclasses for distinct CSML coverage geometries

5.2 CSML coverage types

As mentioned above, CSML feature types are distinguished primarily on the geometry of their coverage domain (for reasons discussed in section 1.3.1). Figure 26 illustrates the modelling of these coverages, and shows the relationship to ISO 19123 coverage classes. CSML coverages are regarded as implementations of the ISO coverage classes.

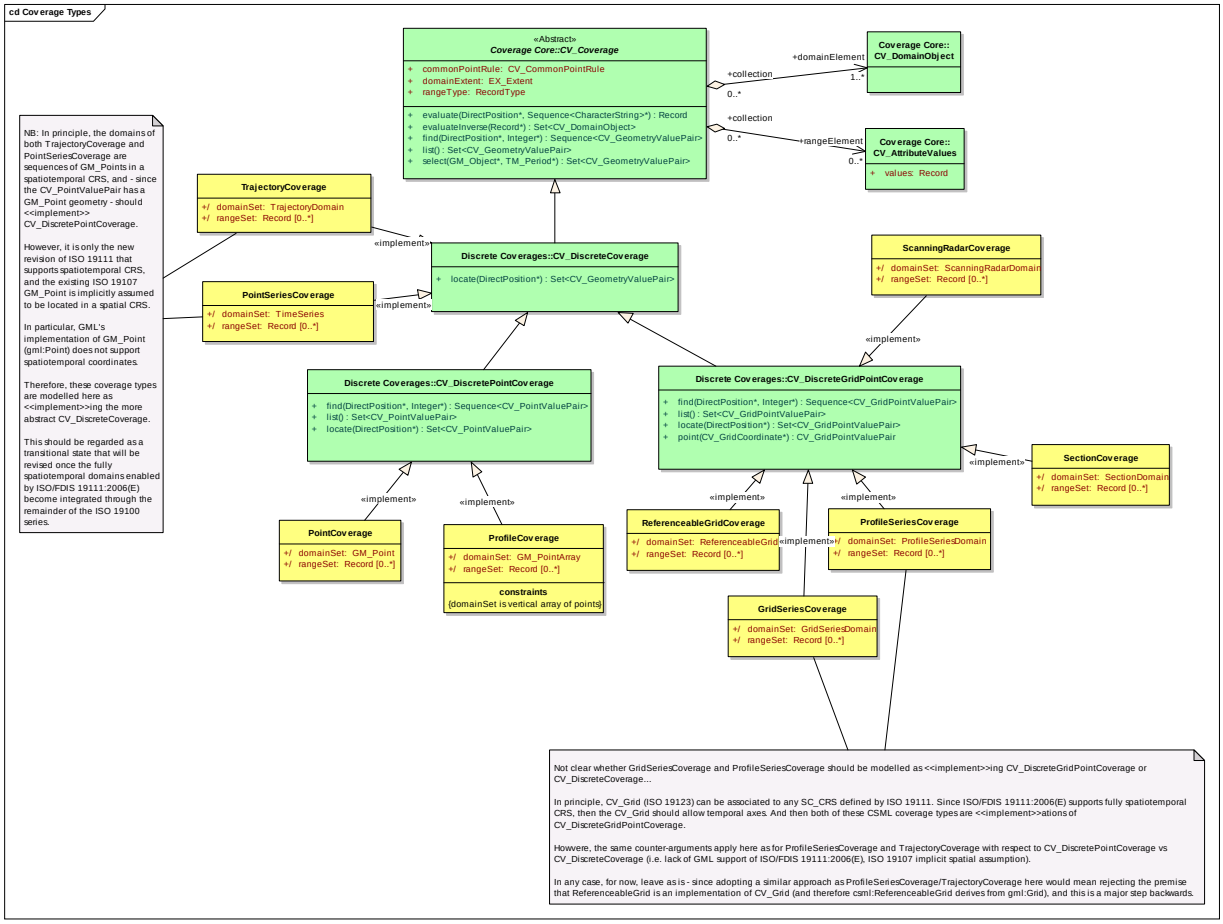

cd Coverage Types
«Abstract»
Coverage Core::CV_Coverage
+ commonPointRule: CV_CommonPointRule
+ domainExtent: EX_Extent
+ rangeType: RecordType
+ evaluate(DirectPosition*, Sequence<CharacterString>*) : Record
+ evaluateInverse(Record*) : Set<CV_DomainObject>
+ find(DirectPosition*, Integer*) : Sequence<CV_GeometryValuePair>
+ list() : Set<CV_GeometryValuePair>
+ select(GM_Object*, TM_Period*) : Set<CV_GeometryValuePair>
+collection
0..*
+domainElement
1..*
Coverage Core::
CV_DomainObject
+collection
0..*
+rangeElement
0..*
Coverage Core::
CV_AttributeValues
+ values: Record
NB: In principle, the domains of both TrajectoryCoverage and PointSeriesCoverage are sequences of GM_Points in a spatiotemporal CRS, and - since the CV_PointValuePair has a GM_Point geometry - should <<implement>> CV_DiscretePointCoverage.
However, it is only the new revison of ISO 19111 that supports spatiotemporal CRS, and the existing ISO 19107 GM_Point is implicitly assumed to be located in a spatial CRS.
In particular, GML's implementation of GM_Point (gml:Point) does not support spatiotemporal coordinates.
Therefore, these coverage types are modelled here as <<implement>>ing the more abstract CV_DiscreteCoverage.
This should be regarded as a transitional state that will be revised once the fully spatiotemporal domains enabled by ISO/FDIS 19111:2006(E) become integrated through the remainder of the ISO 19100 series.
TrajectoryCoverage
+/ domainSet: TrajectoryDomain
+/ rangeSet: Record [0..*]
«implement»
PointSeriesCoverage
+/ domainSet: TimeSeries
+/ rangeSet: Record [0..*]
«implement»
Discrete Coverages::CV_DiscreteCoverage
+ locate(DirectPosition*) : Set<CV_GeometryValuePair>
ScanningRadarCoverage
+/ domainSet: ScanningRadarDomain
+/ rangeSet: Record [0..*]
«implement»
Discrete Coverages::CV_DiscretePointCoverage
+ find(DirectPosition*, Integer*) : Sequence<CV_PointValuePair>
+ list() : Set<CV_PointValuePair>
+ locate(DirectPosition*) : Set<CV_PointValuePair>
Discrete Coverages::CV_DiscreteGridPointCoverage
+ find(DirectPosition*, Integer*) : Sequence<CV_GridPointValuePair>
+ list() : Set<CV_GridPointValuePair>
+ locate(DirectPosition*) : Set<CV_GridPointValuePair>
+ point(CV_GridCoordinate*) : CV_GridPointValuePair
«implement»
SectionCoverage
+/ domainSet: SectionDomain
+/ rangeSet: Record [0..*]
«implement»
«implement»
«implement»
«implement»
PointCoverage
+/ domainSet: GM_Point
+/ rangeSet: Record [0..*]
«implement»
ProfileCoverage
+/ domainSet: GM_PointArray
+/ rangeSet: Record [0..*]
constraints
{domainSet is vertical array of points}
«implement»
ReferenceableGridCoverage
+/ domainSet: ReferenceableGrid
+/ rangeSet: Record [0..*]
«implement»
ProfileSeriesCoverage
+/ domainSet: ProfileSeriesDomain
+/ rangeSet: Record [0..*]
GridSeriesCoverage
+/ domainSet: GridSeriesDomain
+/ rangeSet: Record [0..*]
Not clear whether GridSeriesCoverage and ProfileSeriesCoverage should be modelled as <<implement>>ing CV_DiscreteGridPointCoverage or CV_DiscreteCoverage...
In principle, CV_Grid (ISO 19123) can be associated to any SC_CRS defined by ISO 19111. Since ISO/FDIS 19111:2006(E) supports fully spatiotemporal CRS, then the CV_Grid should allow temporal axes. And then both of these CSML coverage types are <<implement>>ations of CV_DiscreteGridPointCoverage.
Howvere, the same counter-arguments apply here as for ProfileSeriesCoverage and TrajectoryCoverage with respect to CV_DiscretePointCoverage vs CV_DiscreteCoverage (i.e. lack of GML support of ISO/FDIS 19111:2006(E), ISO 19107 implicit spatial assumption).
In any case, for now, leave as is - since adopting a similar approach as ProfileSeriesCoverage/TrajectoryCoverage here would mean rejecting the premise that ReferenceableGrid is an implementation of CV_Grid (and therefore csml:ReferenceableGrid derives from gml:Grid), and this is a major step backwards.

Figure 26: CSML coverage classes

5.3 CSML Feature Types

We now consider in turn the respective CSML feature types.

5.3.1 PointFeature

The CSML PointFeature represents a single point measurement at a fixed position in space (and optionally time), and is modelled as shown in Figure 27.

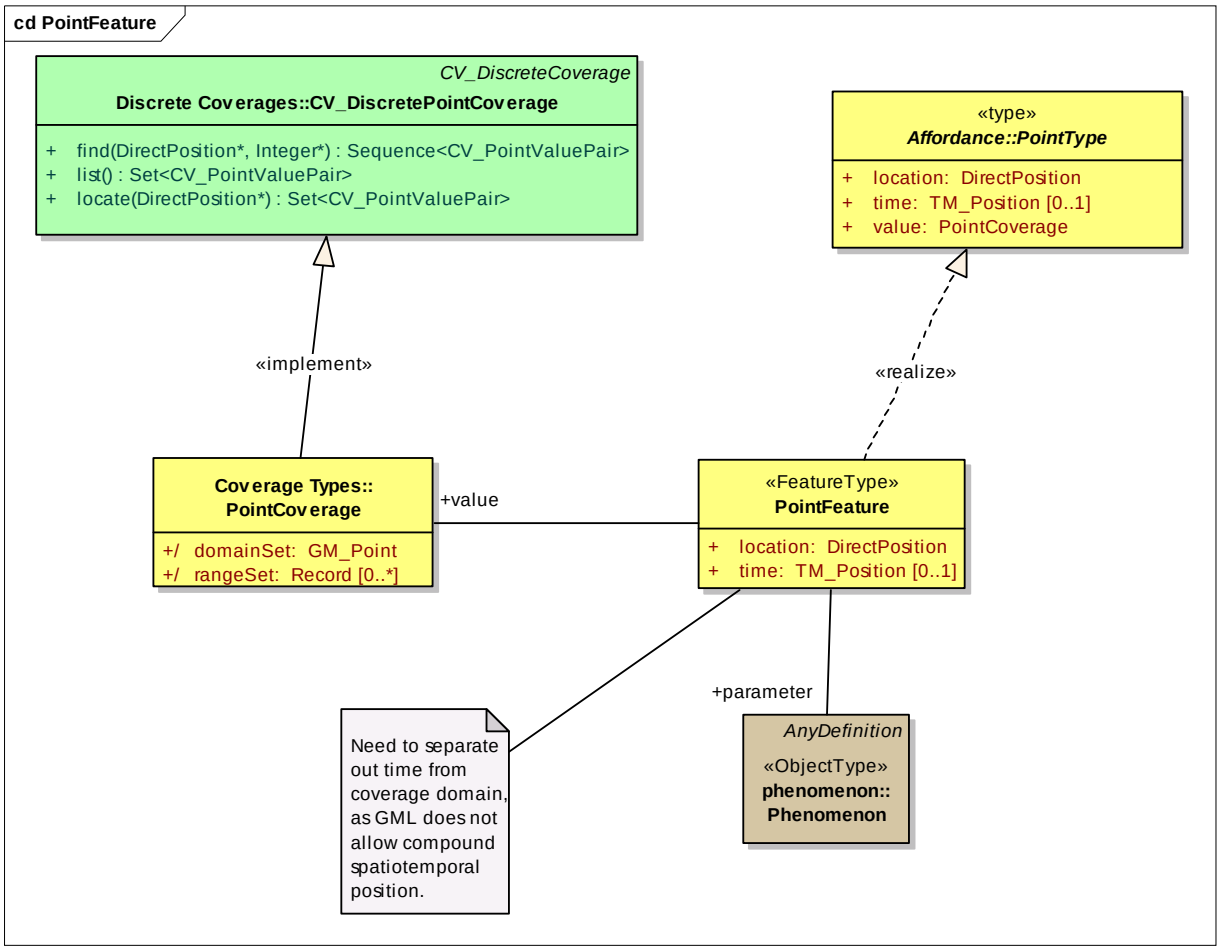


Figure 27: CSML PointFeature - UML model

The attributes are:

- **location**: spatial location of the point measurement
- **time**: (optional) time of the point measurement

The coverage domain consists of the single point location of the measurement.

also want: illustrative picture, csml example, conventional portrayal

5.3.2 *PointSeriesFeature*

The CSML PointSeriesFeature represents a time-series of single datum measurements at a fixed location in space.

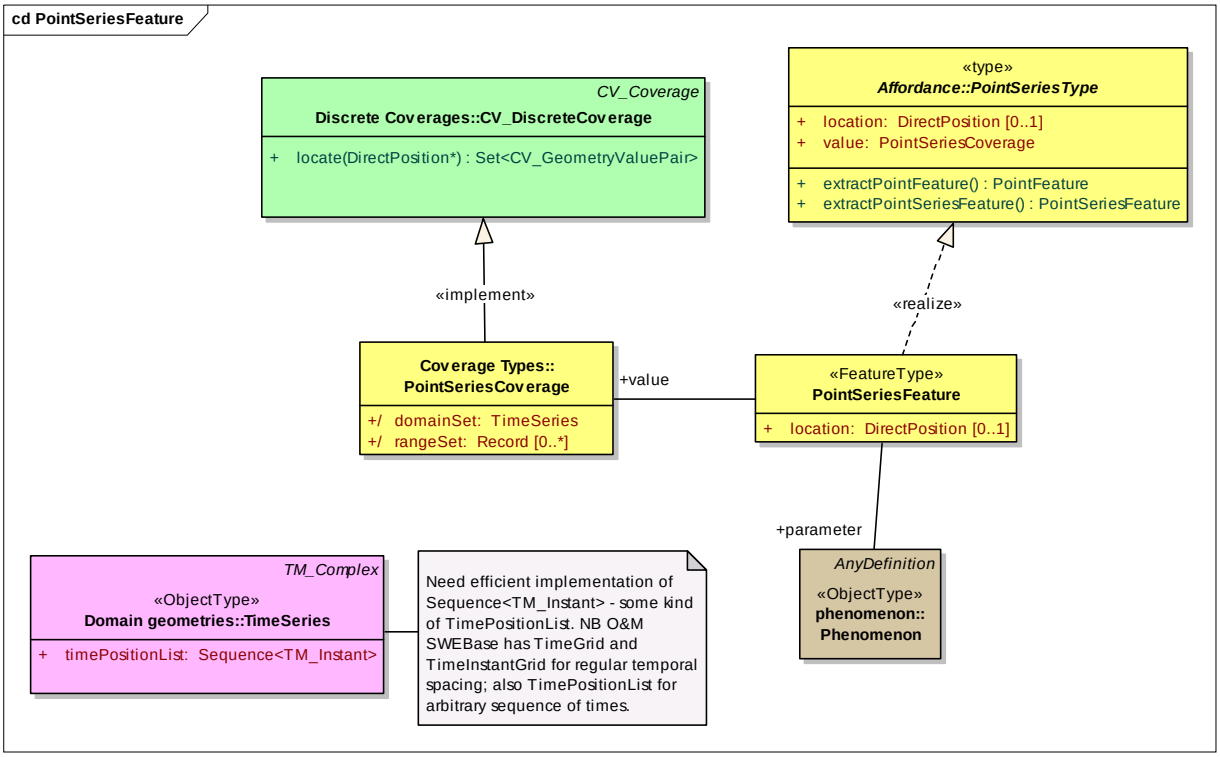


Figure 28: CSML PointSeriesFeature - UML model

The attributes are:

- **location**: the spatial location from which the profile was measured.

The coverage domain consists of a series of time instants (at which values are recorded).

5.3.3 *TrajectoryFeature*

The TrajectoryFeature provides a measurement along a discrete path in time and space.

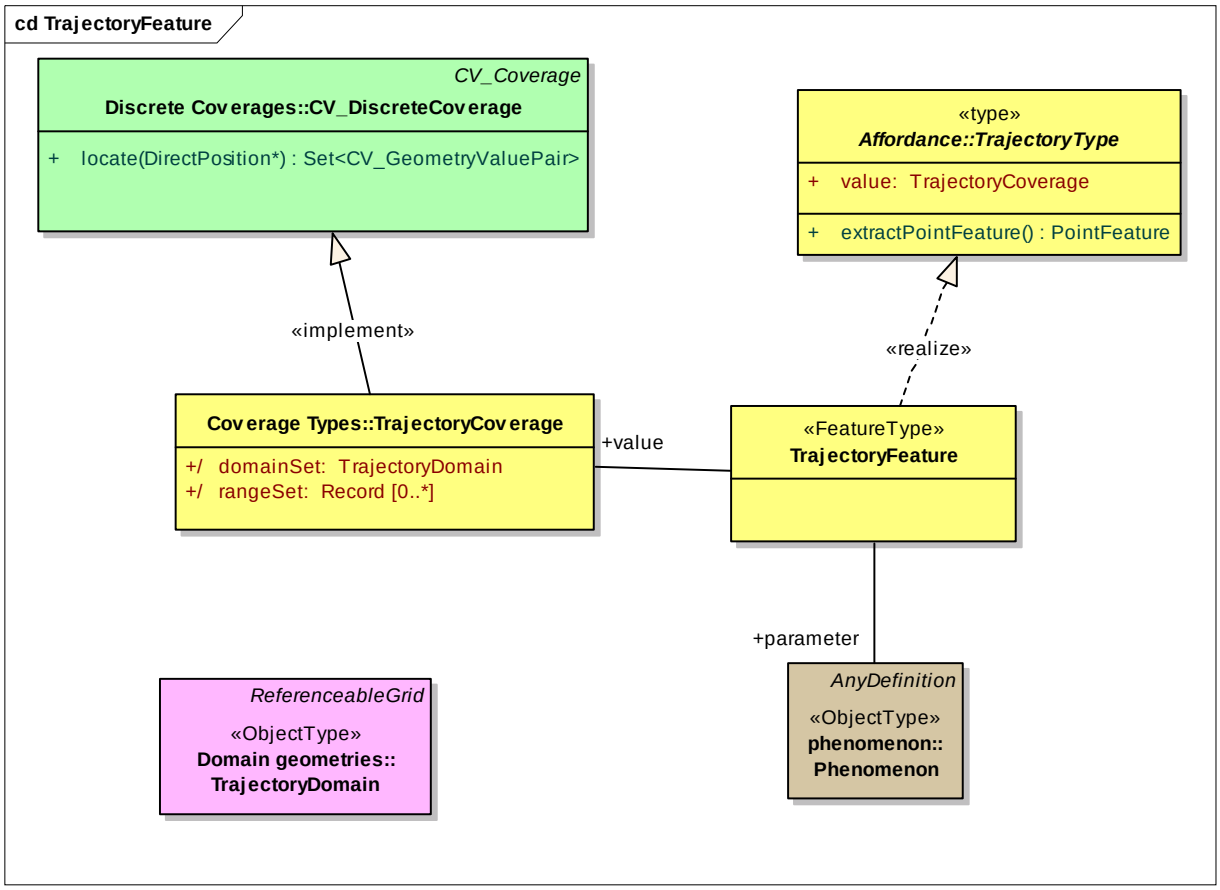


Figure 29: CSML TrajectoryFeature - UML model

There are no intrinsic attributes of the feature type, but its associated coverage is defined over a TrajectoryDomain – a one-dimensional irregular grid embedded within a 2-, 3-, or 4-d compound spatiotemporal coordinate reference system (Figure 25). Note: the reason for using a subclass of ReferenceableGrid for the domain – rather than simply using the gml:MultiPointCoverage, is that the former supports full spatiotemporal geometry.

5.3.4 PointCollectionFeature

A PointCollectionFeature is a collection of distributed single datum measurements at a particular time.

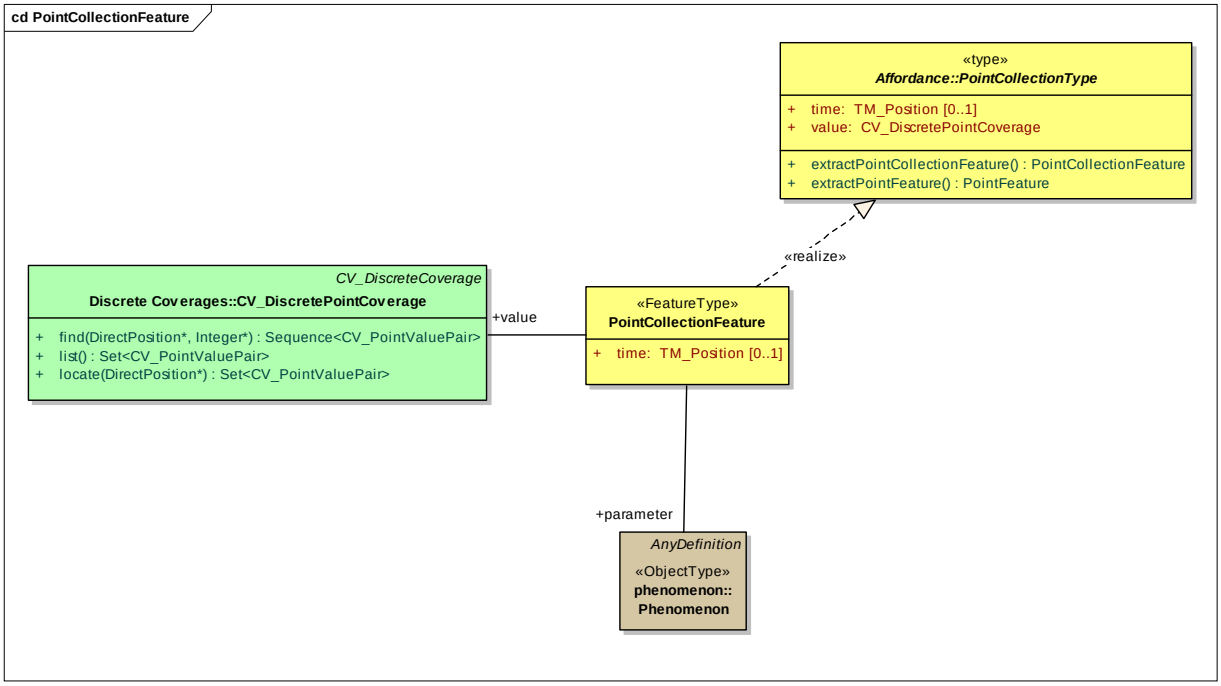


Figure 30: CSML PointCollectionFeature - UML model

The sole attribute is:

- **time**: (optional) the time at which the collection of measurements was taken

The associated coverage provides the values at the set of measurement locations, and is exactly an ISO 19123 CV_DiscretePointCoverage (with gml:MultiPointCoverage used for encoding).

5.3.5 *ProfileFeature*

A ProfileFeature provides a single 'profile' of some parameter along a vertical line in space.

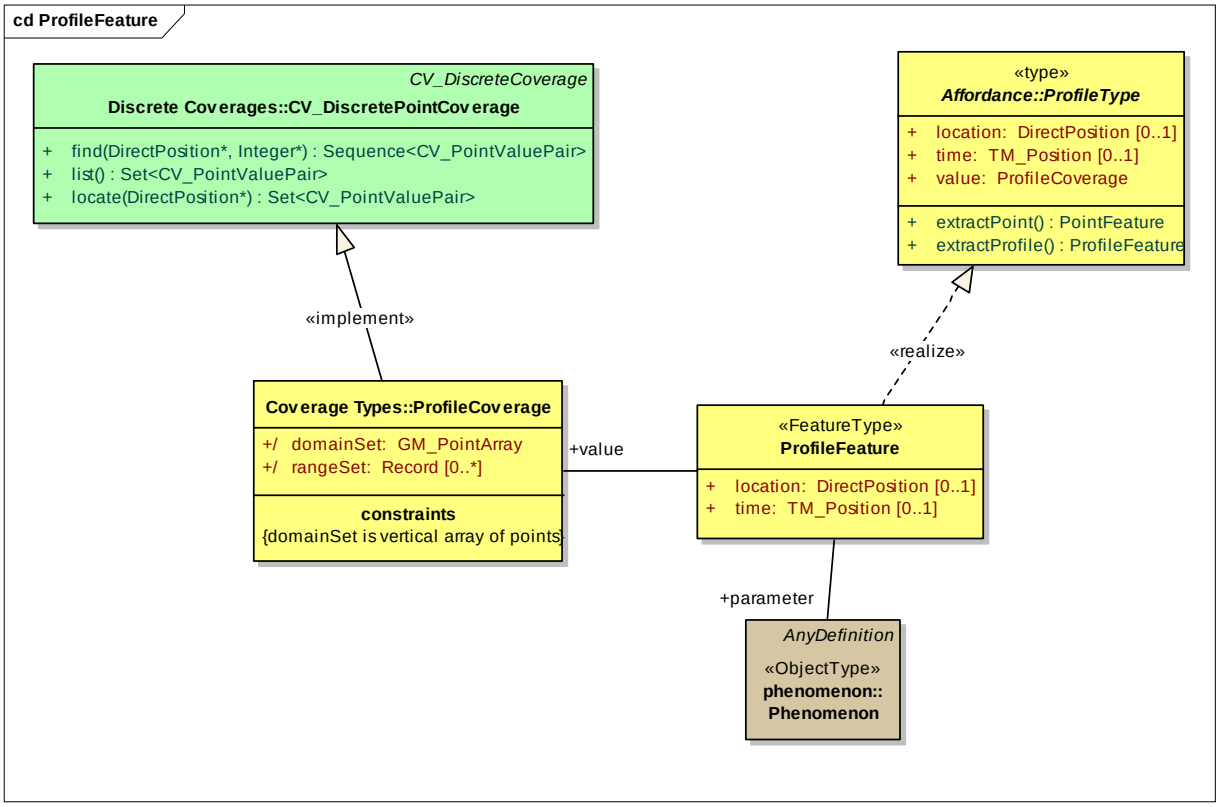


Figure 31: CSML ProfileFeature - UML model

The attributes of this feature type are:

- **location**: (optional) the base horizontal location from which the profile measurements were taken
- **time**: (optional) the nominal time at which the profile measurements were made

The associated coverage provides values over the set of vertical levels. A CSML ProfileCoverage is used, which is simply an ISO 19123 CV_DiscretePointCoverage (gml:MultiPointCoverage) with its domain restricted to a vertical array of points.

5.3.6 *ProfileSeriesFeature*

A ProfileSeriesFeature represents a time-series of profiles on fixed vertical levels at a fixed location.

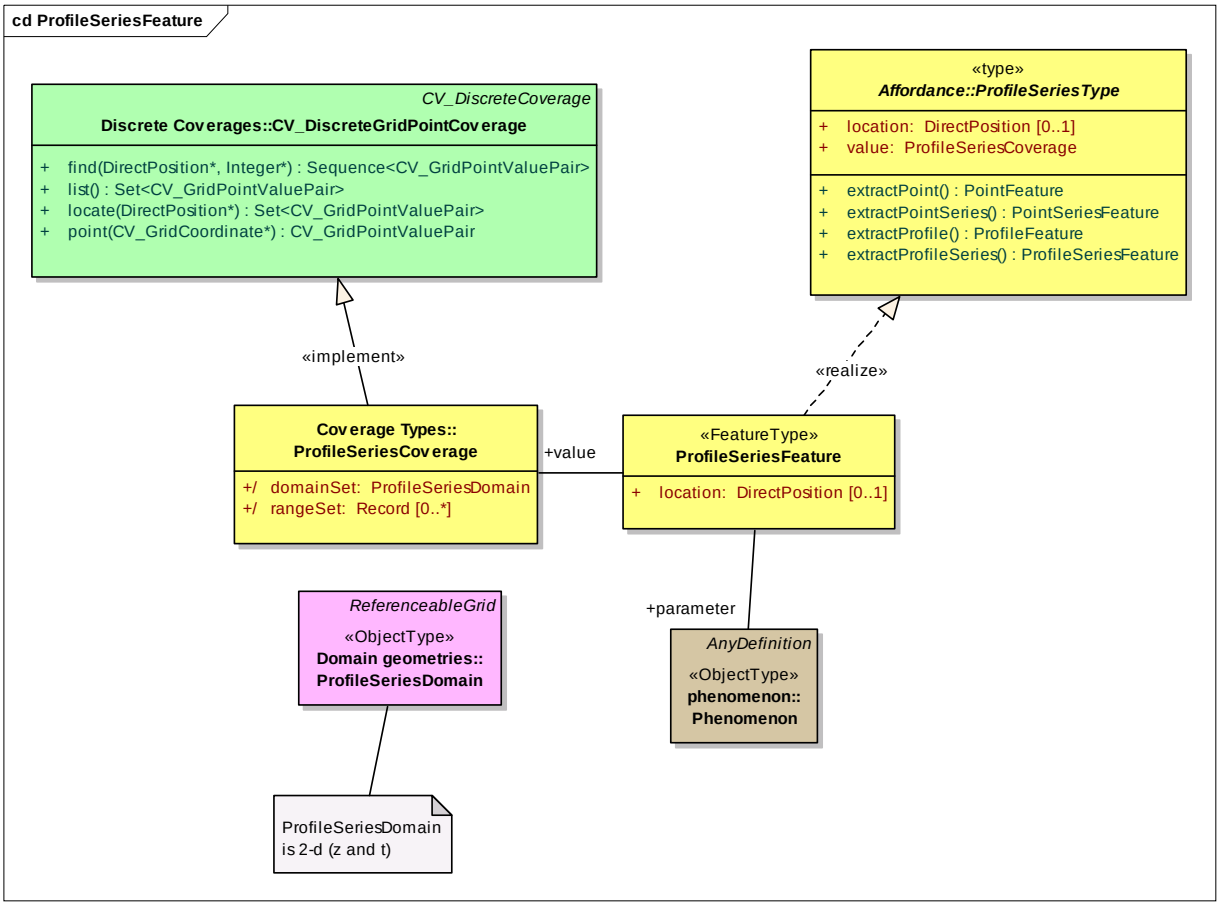


Figure 32: CSML ProfileSeriesFeature - UML model

The feature type has a single attribute:

- **location**: (optional) the horizontal location at which the profiles were measured

The profile values are provided as a gridded coverage (ProfileSeriesCoverage) over a two-dimensional height-time spatiotemporal domain (ProfileSeriesDomain, specialising ReferenceableGrid).

5.3.7 RaggedProfileSeriesFeature

A RaggedProfileSeriesFeature is a time-series of unequal-length profiles, but on fixed vertical levels, at a fixed location.

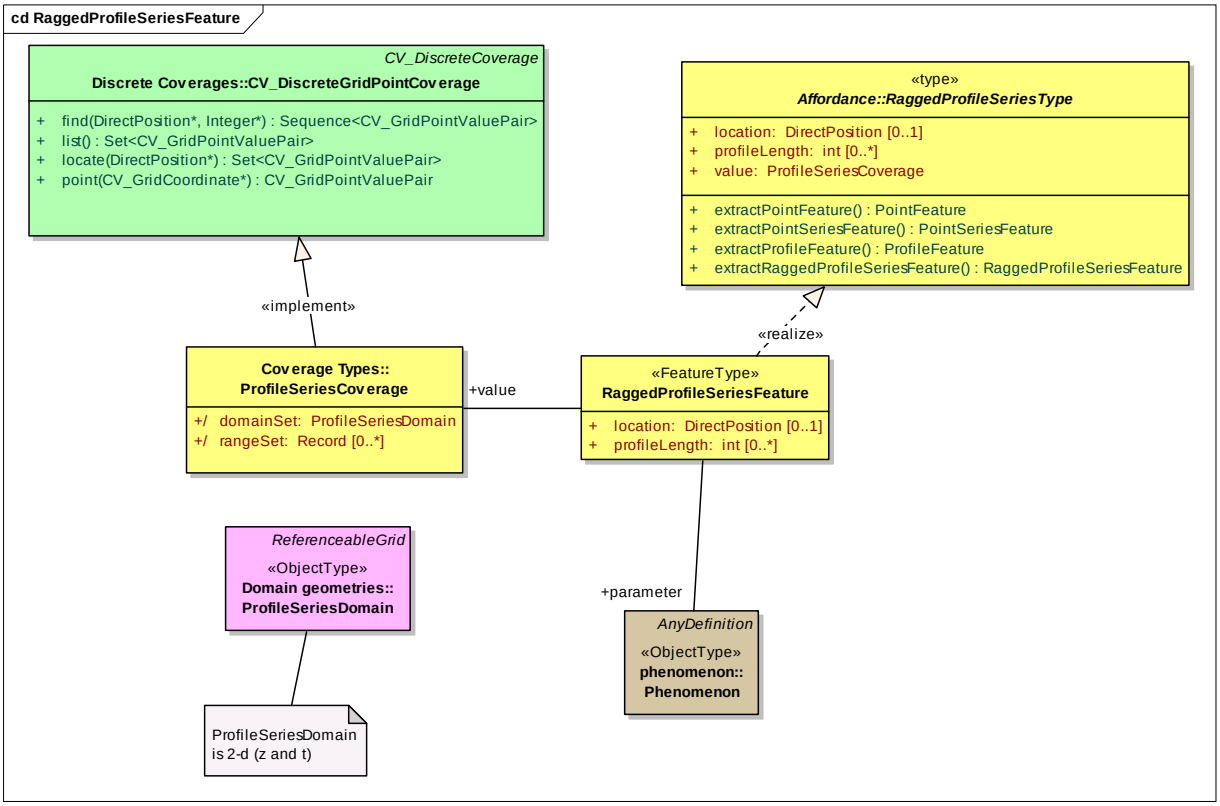


Figure 33: CSML RaggedProfileSeriesFeature - UML model

There are two attributes:

- **location**: (optional) the horizontal location at which the profiles were measured
- **profileLength**: the number of valid measurements per profile for the series

The measured profile values are provided through a ProfileSeriesCoverage, but where values at missing vertical levels on each profile are ignored. It is assumed implicitly that these missing values occur at the extreme end of the profile in each case[4] (i.e. the seafloor end of a marine profile taken from the surface, or the up-most height levels for an atmospheric sounding from the earth). The motivation for the use of a ProfileSeriesCoverage for this purpose is not cleanly conceptual, but has a significant encoding efficiency benefit – i.e. repeating vertical measurement locations don't need to be repeated in the encoding.

[4] An alternative approach could specify the start and end index (rather than total profile length) for each profile in the series.

5.3.8 SectionFeature

A SectionFeature provides a series of profiles from positions along a trajectory in time and space.

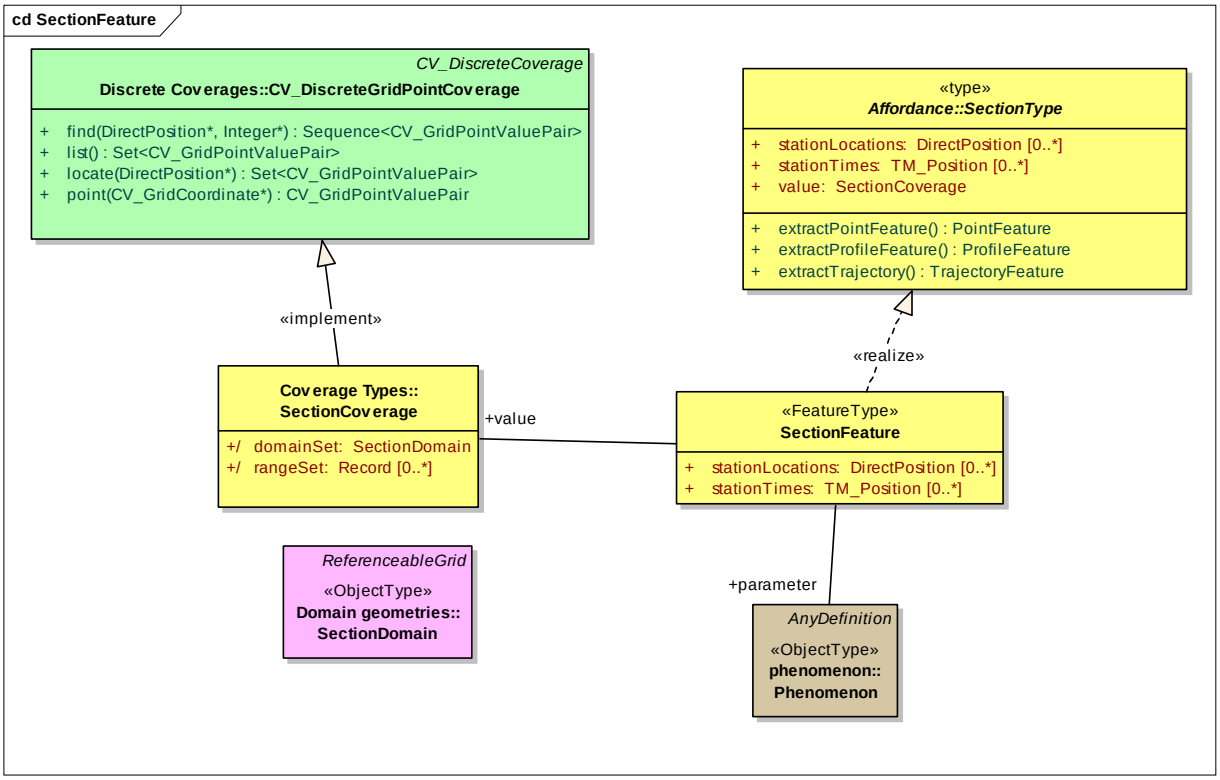


Figure 34: CSML SectionFeature - UML model

The following attributes are defined:

- **stationLocations**: (optional) an array of locations along the trajectory at which profile measurements are taken
- **stationTimes**: (optional) the list of times at which the profiles were taken

The values along the section are available in a CSML SectionCoverage, with a domain that is a two-dimensional grid embedded in three-dimensional space and with one axis aligned vertically (Figure 25).

5.3.9 *RaggedSectionFeature*

A RaggedSectionFeature is a series of profiles of unequal length along a trajectory in time and space.

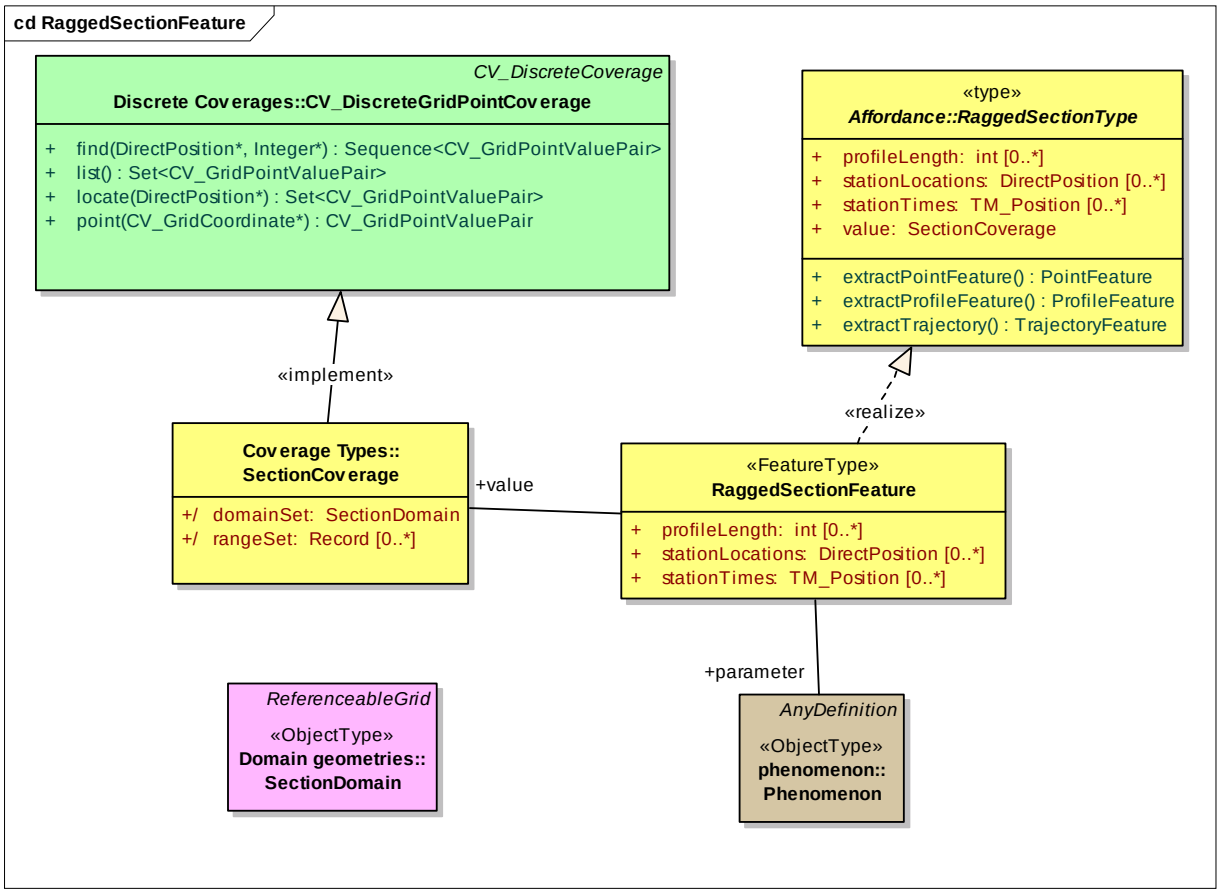


Figure 35: CSML RaggedSectionFeature- UML model

It has the following attributes:

- **stationLocations**: (optional) an array of locations along the trajectory at which profile measurements are taken
- **stationTimes**: (optional) the list of times at which the profiles were taken
- **profileLength**: the number of valid measurements per profile for the series

The measured profile values along the section are provided through a SectionCoverage, but where values at missing vertical levels on each profile are ignored. It is assumed implicitly that these missing values occur at the extreme end of the profile in each case[5] (i.e. the seafloor end of a marine profile taken from the surface, or the up-most height levels for an atmospheric sounding from the earth).

[5] An alternative approach could specify the start and end index (rather than total profile length) for each profile in the series.

5.3.10 ScanningRadarFeature

A ScanningRadarFeature provides backscatter profiles along a look direction at fixed elevation but rotating in azimuth.

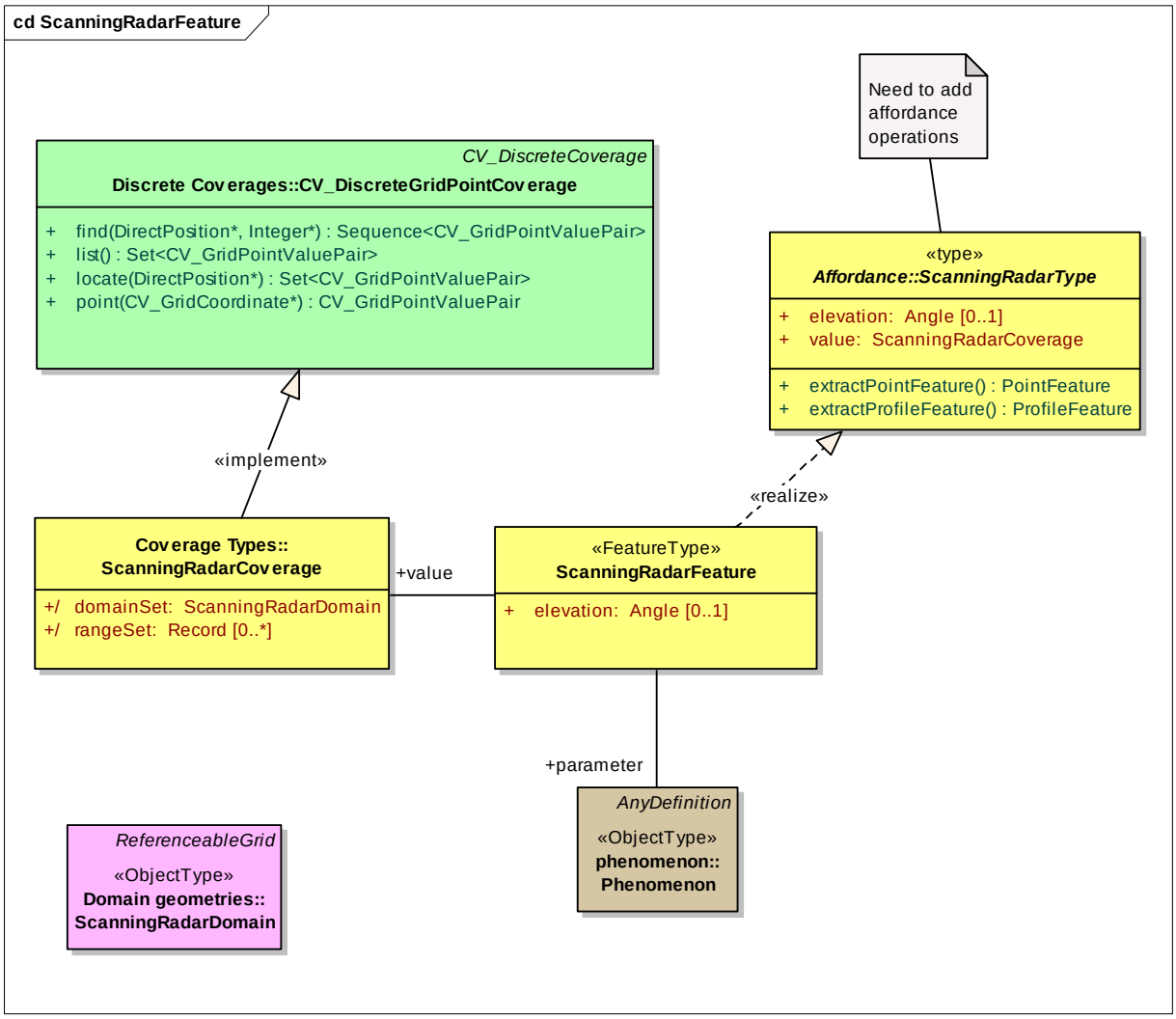


Figure 36: CSML ScanningRadarFeature - UML model

There is one attribute for a ScanningRadarFeature:

- **elevation**: (optional) the fixed elevation of the radar scanning beam

The radar measured values are provided in a ScanningRadarCoverage. This has a domain that is a two-dimensional grid in a coordinate reference system with axes of azimuth and range. Thus, the coverage domain is modelled (Figure 25) as a specialisation (ScanningRadarDomain) of a ReferenceableGrid that must be associated to such a coordinate reference system.

CSML provides the specialised AzimuthRangeCRSType for this purpose, restricting an EngineeringCRS to consist of a polar coordinate system (gml:polarCS) and a specialised engineering datum, the CSML azimuthRangeDatum.

5.3.11 GridFeature

A CSML GridFeature represents a single time-snapshot of a gridded field.

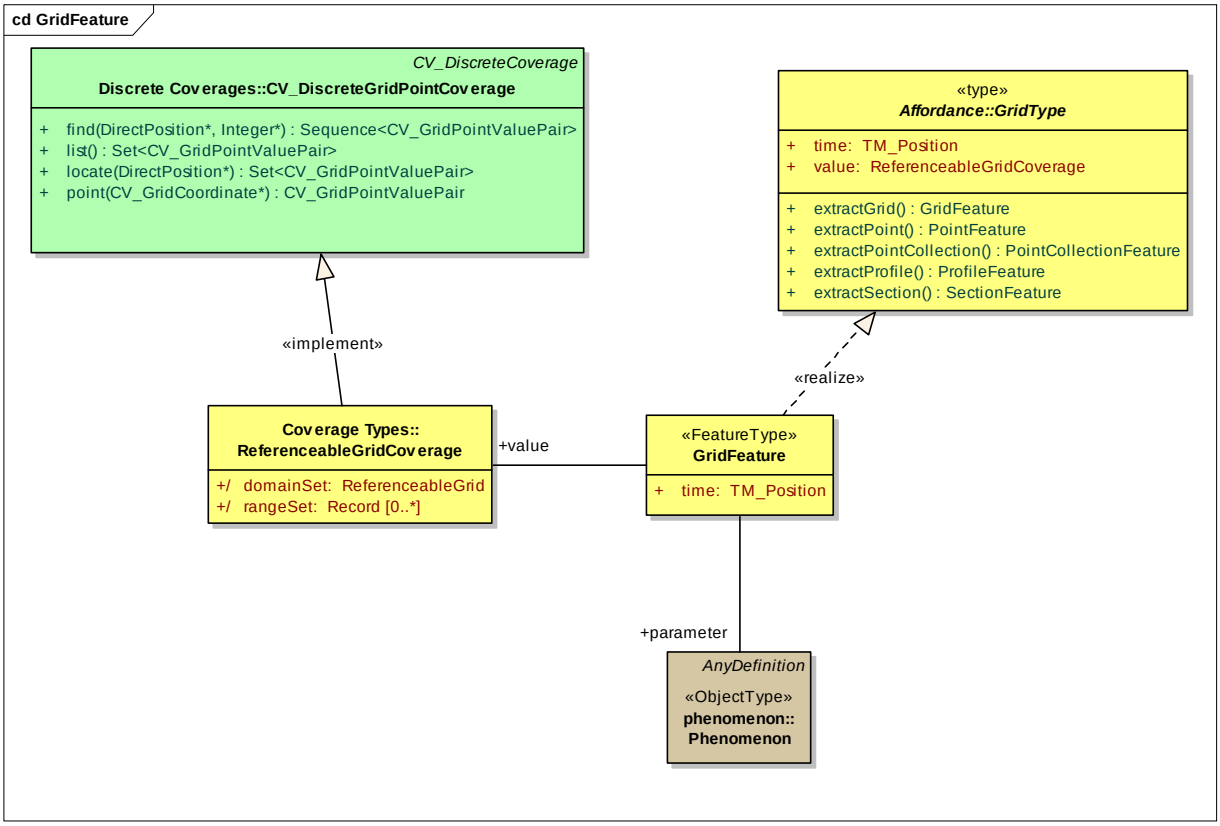


Figure 37: CSML GridFeature - UML model

The single attribute is:

- **time**: the time of the gridded field

The field values are provided as a coverage (ReferenceableGridCoverage) over an irregular grid (of any dimension, embedded within a coordinate space of any dimension). It uses directly the ReferenceableGrid structure for the domain geometry.

5.3.12 GridSeriesFeature

A GridSeriesFeature represents a time-series of gridded parameter fields.

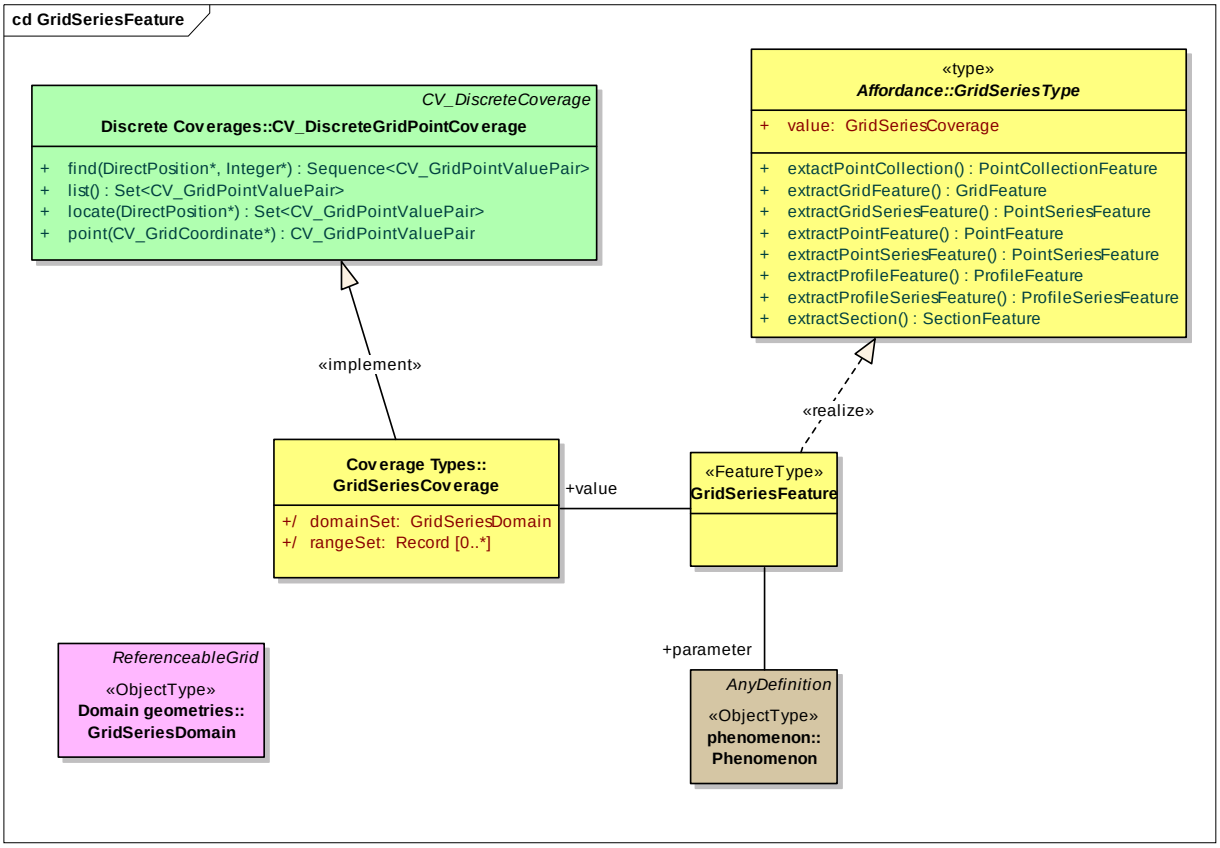


Figure 38: CSML GridSeriesFeature - UML model

The feature type has no particular attributes. The gridded time-series of values are provided through a three- or four-dimensional gridded coverage under an associated spatiotemporal compound coordinate reference system (Figure 25).

5.3.13 SwathFeature

The CSML SwathFeature represents a two-dimensional grid of data along a satellite ground-path.

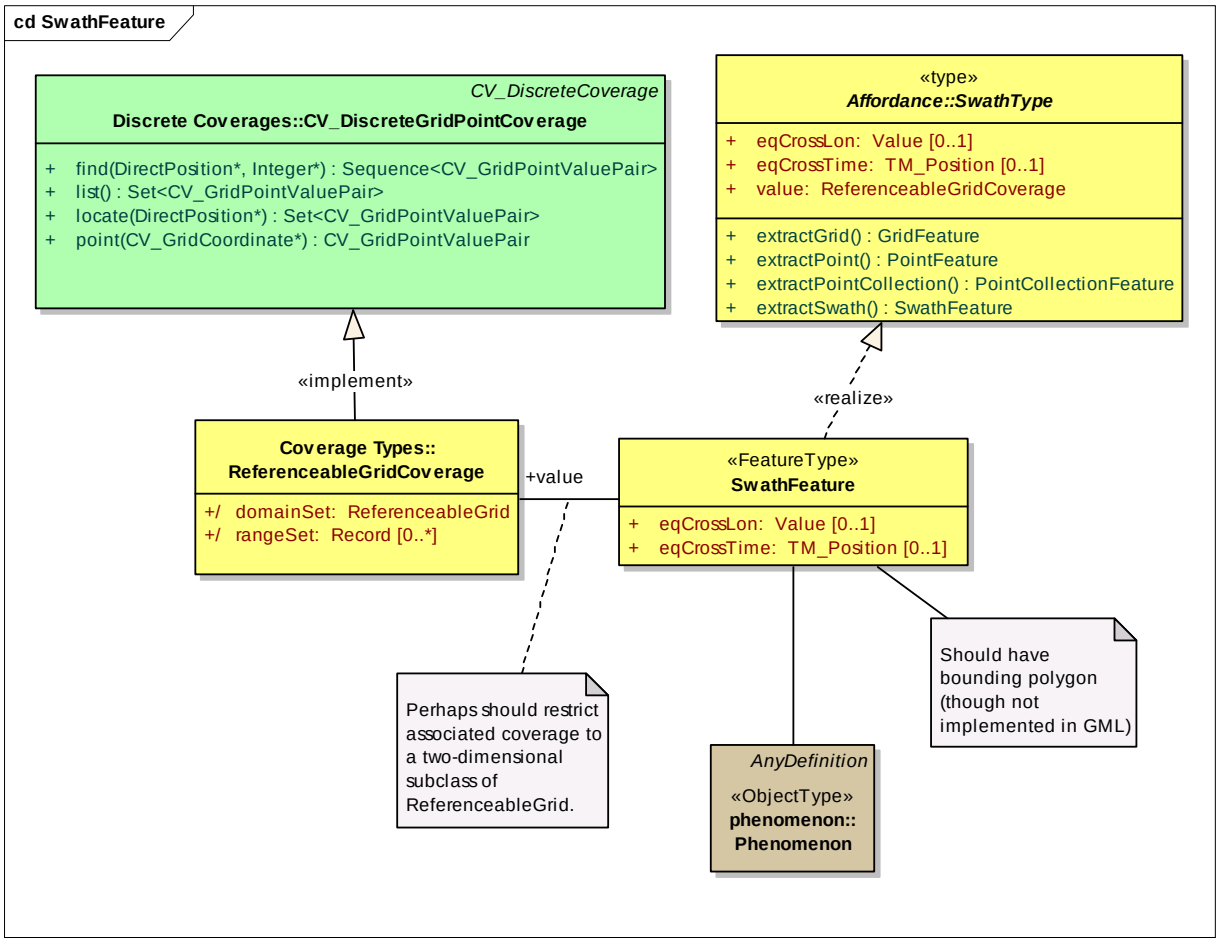


Figure 39: CSML SwathFeature - UML model

The feature type attributes are:

- **eqCrossLon**: (optional) the equator crossing longitude for the satellite pass
- **eqCrossTime**: (optional) the equator crossing time for the satellite pass

The observed swath values are provided through a two-dimensional gridded coverage over the earths' surface – the ReferenceableGrid is used for the domain.

6. FILE INTEGRATION

The mechanism for integrating file-based data in CSML was outlined in section 1.2. There are two components to the approach: (1) a conceptual model of feature types is defined and feature instances are encoded using GML (omitting any content that conceptually comes from files), (2) a 'storage descriptor' schema is used to describe logical extraction of data from storage. The mapping of the latter into the former is achieved using a specific xlink pattern, described in section 6.2 below.

The CSML feature collection and storage descriptors exist in two logically separate XML documents, but for convenience they may be grouped together using the CSML 'Dataset' element as described in section 3.3.

This chapter describes in detail the CSML storage descriptor and the xlink mechanism.

6.1 CSML StorageDescriptor

The conceptual model for the CSML StorageDescriptor is shown in Figure 40.

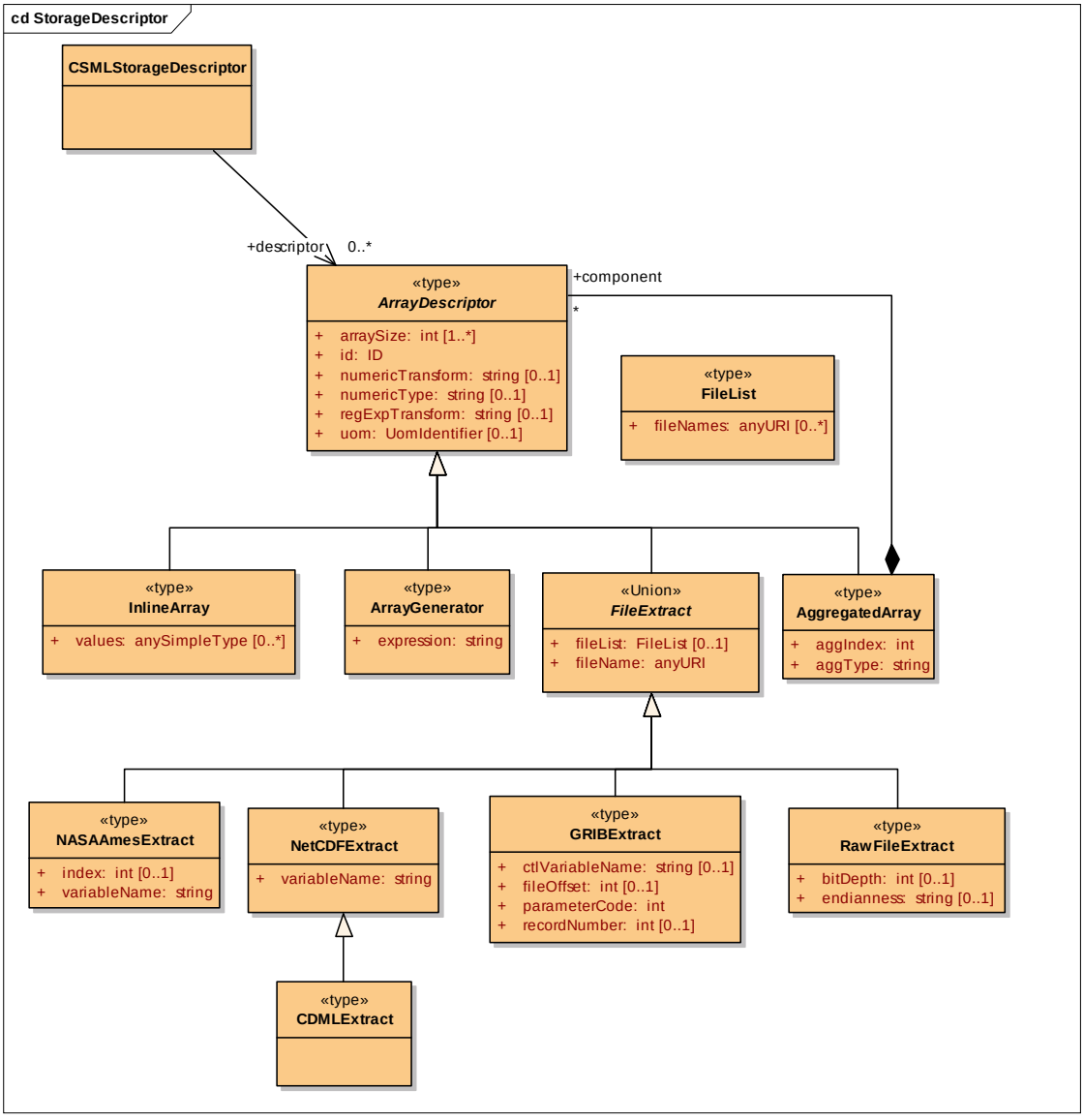


Figure 40: CSML numeric array definition

The following four array descriptor subclasses are defined:

- InlineArray: an inline character encoding of array values
- ArrayGenerator: a formulaic expression implicitly generating array values
- FileExtract: values extracted from file-based storage
- AggregatedArray: an aggregation of two or more arrays along an existing or new dimension

In addition a numeric or regular expression transformation can be applied to any of the above as appropriate.

6.1.1 ArrayDescriptor

This is the abstract base class for all CSML numeric array descriptors. The following attributes are defined, and inherited by all concrete classes:

- **arraySize**: a sequence of integers specifying the dimensions of the array being defined.
- **uom**: units of measure. This is required whenever the array descriptor is used for a coverage rangeSet (when used for the domainSet, the coordinate axis units are specified in the parent geometry object).
- **numericType**: a string ('float', 'double', 'short', 'int') defining the numeric type to which the values should be cast in a user application.
- **numericTransform**: an expression providing a mathematical transformation to be applied to the numeric values defined by this descriptor.
- **regExpTransform**: a regular expression providing a textual transformation to be applied to the direct character-based data defined by this descriptor (e.g. where inline values are supplied, or the file is ASCII-encoded). This is applied before any cast to a numerical type within the user application

6.1.2 InlineArray

This class is used for specifying values as direct inline XML content. The only attribute is:

- **values**: a list of values (any simple type)

An example XML encoding is shown in Figure 41. This example utilises all the optional attributes, and demonstrates both the regular expression and numeric transformation. The result of this array descriptor is a five-by-two array of temperatures ranging from 6 to 14 with the final two values equal.

```
<InlineArray>
    <arraySize>5 2</arraySize>
    <uom>udunits.xml#degreeC</uom>
    <numericType>float</numericType>
    <regExpTransform>s/10/9/ge</regExpTransform>
    <numericTransform>+5</numericTransform>
    <values>1 2 3 4 5 6 7 8 9 10</values>
```

```
</InlineArray>
```

Figure 41: InlineArray for value array.

*[***TBD: need to define precise syntax and semantics for numericTransform and regExpTransform (use flexible perl regexp model)]*

6.1.3 ArrayGenerator

If the numeric values in an array follow a simple pattern, it is more efficient to encode them implicitly through a formulaic expression. The ArrayGenerator class provides this facility. One attribute is defined:

- **expression**: provides a formulaic expression defining values to be generated implicitly

An example XML encoding is shown in Figure 42. This example generates a sequence of 10001 values representing five minute increments in time from zero.

```
<ArrayGenerator>
    <arraySize>10001</arraySize>
    <uom>udunits.xml#minute</uom>
    <numericType>float</numericType>
    <expression>0:5:50000</expression>
</ArrayGenerator>
```

Figure 42: XML encoding example for ArrayGenerator.

*[***TBD: Need to define precise syntax and semantics of expression. Perhaps based on simple MATLAB-like multi-d array expressions? NB: There are use cases where a parametrised expression is needed, with the parameters coming from external storage (e.g. model sigma levels defined by coefficients in a file)]*

6.1.4 FileExtract

Numeric values may be defined in file-based storage. The FileExtract and its subclasses are used for this purpose. The following attribute is defined:

- **fileName**: a pathname to the file.

The class is subclassed for NASA Ames, netCDF, GRIB and CDML files.

6.1.4.1 NASAAmesExtract

A NASAAmesExtract is used where data is being extracted from a NASA Ames format file. The following attributes provide the minimal information required to identify the data to be extracted:

- **variableName**: the name of the variable in the NASA Ames file
- **index**: there is no requirement for NASA Ames variable names to be unique. In the event of duplication, the "index" attribute allows a specific instance to be identified

An example is given in Figure 43.

```
<NASAAmesExtract>
    <arraySize>526</arraySize>
    <numericType>double</numericType>
    <fileName>/data/BADC/macehead/mh960607.asv</fileName>
    <variableName>coefficient a1</variableName>
</NASAAmesExtract>
```

Figure 43: Encoding for NASAAmesExtract.

6.1.4.2 NetCDFExtract

Only a variable name attribute is required in addition to the file name to identify an extract of data from a netCDF file:

- **variableName**: the netCDF variable defining the contents of this numerical array

An example is shown in Figure 44.

```
<NetCDFExtract gml:id="feat04azimuth">
    <arraySize>10000</arraySize>
    <fileName>radar_data.nc</fileName>
    <variableName>az</variableName>
</NetCDFExtract>
```

Figure 44: Encoding of NetCDFExtract.

6.1.4.3 GRIBExtract

The GRIBExtract class represents data extracted from a GRIB format file. In order to uniquely identify the data required, the following parameters are used:

- **parameterCode**: the GRIB parameter code for the GRIB record to be extracted
- **recordNumber**: since a GRIB file can contain any number of records of the same parameter, the specific record number may be indicated
- **fileOffset**: as an alternative to specifying the record number, an explicit offset within the file to the start of the required GRIB record may be specified (this avoids having to step through the file over earlier records until the required record is found)

An XML example is shown in Figure 45.

```
<GRIBExtract>
    <arraySize>320 160</arraySize>
    <numericType>double</numericType>
    <fileName>/e40/ggas1992010100rsn.grb</fileName>
    <parameterCode>203</parameterCode>
    <recordNumber>5</ recordNumber>
    <fileOffset>289412</fileOffset>
</GRIBExtract>
```

Figure 45: XML encoding for GRIBExtract.

*[***TBD: Need to think about and say something about thinned grids etc. Initially, probably user application interpolates onto whatever grid size specified in 'arraySize']*

6.1.4.4 CDMLExtract

Conceptually, this is modelled the same as a netCDF file. Only a variable name attribute is required in addition to the file name to identify an extract of data from a file-set described through CDML:

- **variableName**: the CDML 'variable' defining the contents of this numerical array

An example is shown in Figure 44.

```
<CDMLExtract gml:id="feat10gridtemp">
    <arraySize>360 180 10000</arraySize>
    <fileName>era40.cdml</fileName>
```

```
    <variableName>temperature</variableName>
</CDMLExtract>
```

Figure 46: Encoding of CDMLExtract.

6.1.5 AggregatedArray

The composite pattern is applied so that an array descriptor may be an aggregation of other array descriptors. Arrays may be aggregated along either an existing dimension (thus maintaining the dimensionality of the arrays), or along a new dimension (thus creating an (n+1)-dimensional array from n-dimensional arrays). The following properties are used:

- **aggType**: a character string indicating aggregation along either an 'existing' or 'new' array dimension

- **aggIndex**: the dimension index along which aggregation is performed. For example two 100x200 arrays may be aggregated along the existing index '1' to produce a 200x200 array, or along the existing index '2' to produce a 100x400 array. Aggregation along a new dimension '1' would create a 2x100x200 array; along a new dimension '2' would create a 100x2x200 array, etc.

6.2 'xlink' pattern for linking stored data

6.2.1 Introduction to xlink

A common pattern with file-based data (and part of the motivation for CSML) is that semantic information structures of interest often need to be logically assembled within and across files. The structures of interest (or 'feature types' under the ISO framework) don't necessarily correspond directly with the data models or layouts within files. What is required is a different 'view' onto the data than that provided by existing file structures. (Note that this type of integration is provided implicitly in the relational model – queries can select and join data across tables in any way required.)

CSML starts by asking what the information structures of interest are, with no notice taken of the storage mechanisms. These information structures are formalised as feature types, and may be represented as GML instances. However, content (typically numerical) within these feature instances is logically derived from files, but may need to be extracted and aggregated in complex ways in order to map into the feature instance. A simple example is illustrated in Figure 47 where a feature type might represent a global field of sea-surface temperature, but the data either side of the equator is stored in two separate physical files.

The CSML StorageDescriptor described above provides a mechanism for describing the logical extraction and aggregation of data across multiple files. What is needed as well is a mechanism for asserting that a resulting such logical chunk of data logically provides the content for part of an XML feature instance. Fortunately, such a mechanism exists already, although it is poorly tested, best practice is far from resolved, and its use for this purpose was almost certainly never envisaged in GML.

The 'XML Linking Language' (xlink, [xxii]) "... allows elements to be inserted into XML documents in order to create and describe links between resources... It allows XML documents to: assert linking relationships among more than two resources (and) associate metadata with a link." GML explicitly supports the use of xlink in the 'by-reference' pattern for property values ([viii, §7.2.3.1]): "the value of the property is available elsewhere, and is identified by the value of an xlink:href attribute on the property element".

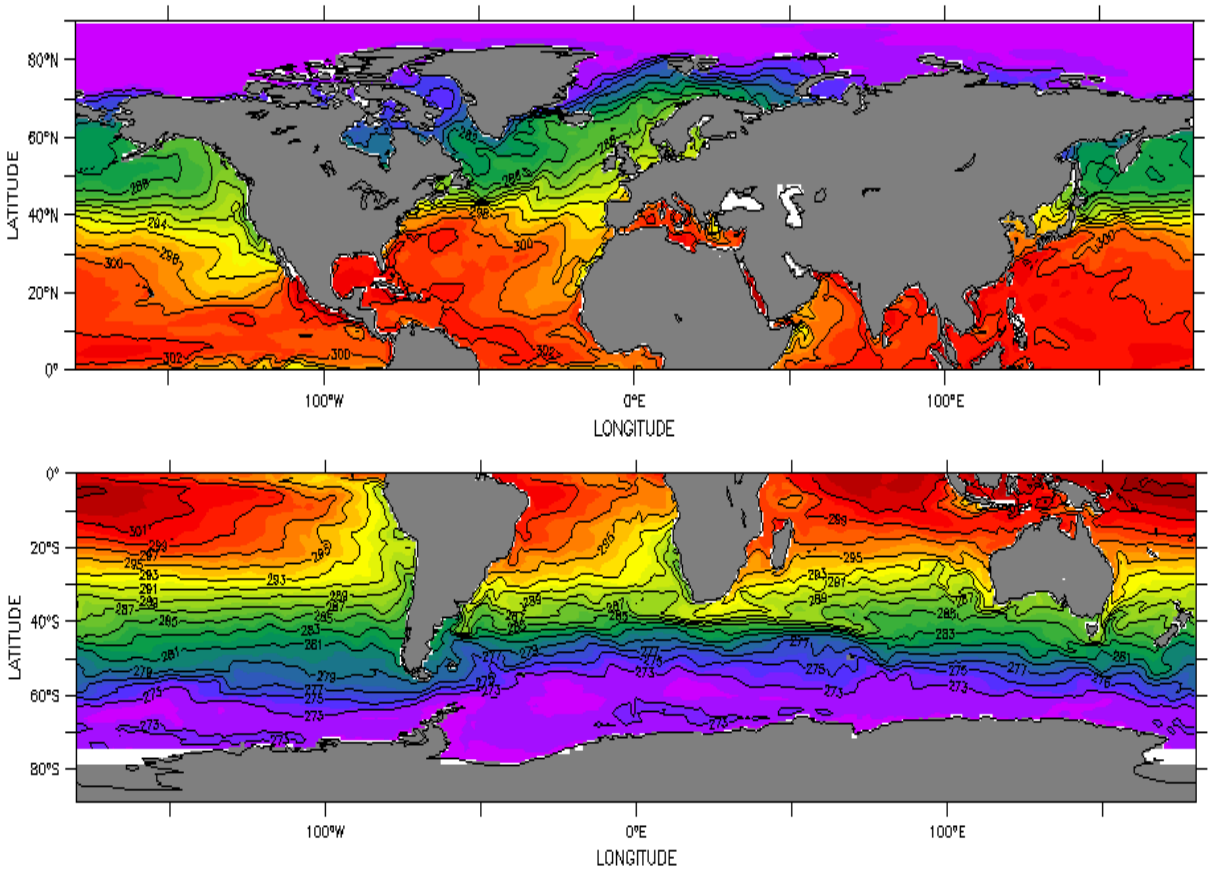


Figure 47: Logically global data stored in separate physical files

The xlink specification describes two models for linking: an 'extended xlink' and a 'simple xlink'. These are illustrated schematically in Figure 48 and Figure 49.

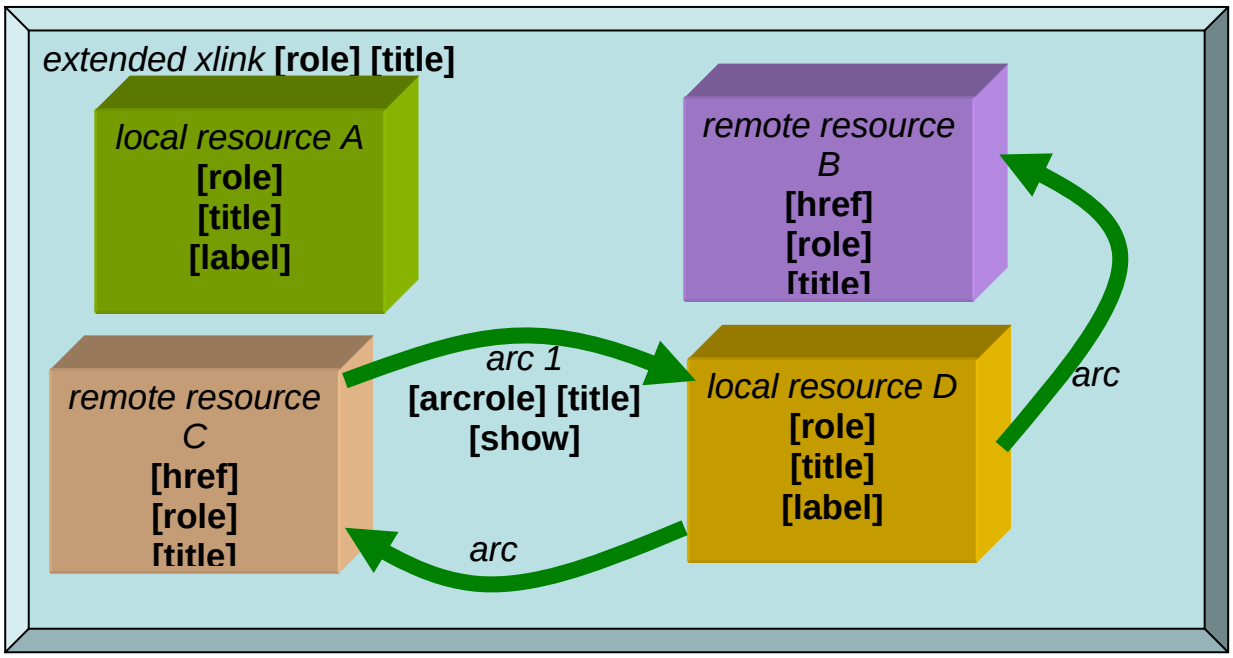


Figure 48: Extended xlink

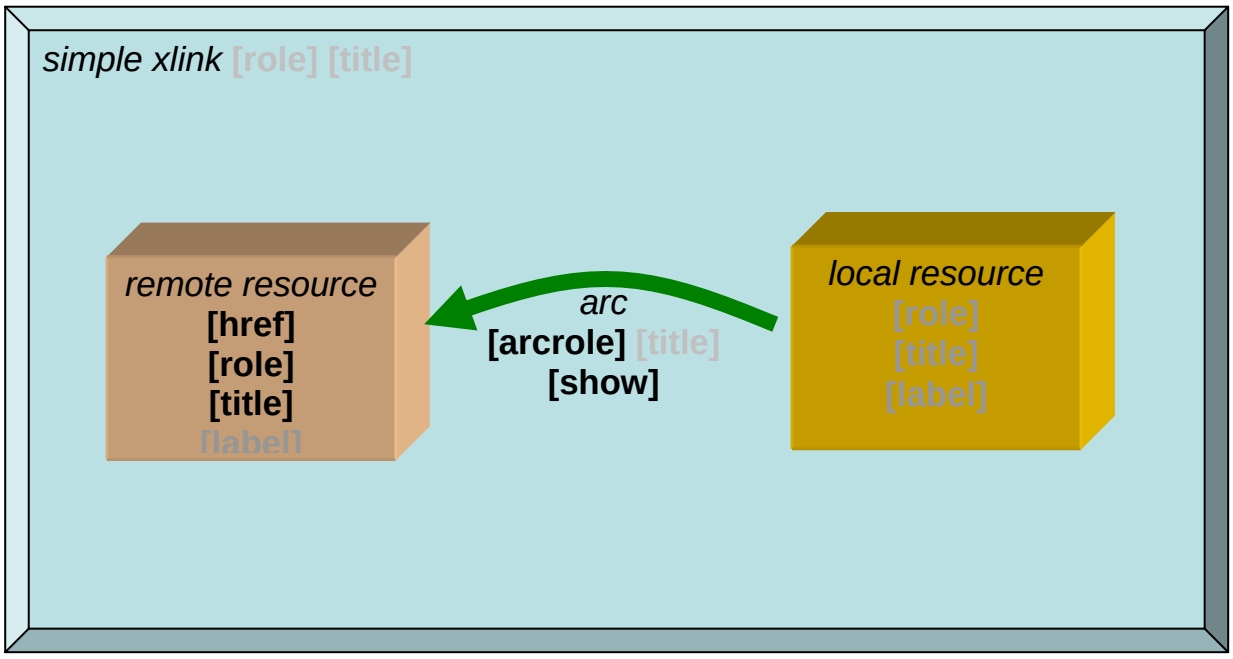


Figure 49: Simple xlink

An extended xlink enables multiple resources to be logically linked together, with numerous metadata assertions concerning the nature of the resources and the links. One model might have been top use such an extended xlink to perform the kinds of multi-file aggregations provided by the CSML StorageDescriptor. However, it is unlikely that the arc traversal rules are sufficiently flexible, and it seems easiest to manage aggregation, parameterised reading of data files, etc. in a separate schema, and define a single clean use of xlink for linking to it.

Thus, we focus on the smaller problem of using a simple xlink to assert that logical content within a GML instance is encapsulated in a CSML ArrayDescriptor. The elements of a simple xlink are as follows (Figure 49):

- **href**: must specify a URI that resolves to a remote resource
- **role**: describes the 'meaning' of the remote resource in the context of the link
- **title**: describes the 'meaning' of the remote resource in a human-readable fashion
- **arcrole**: describes the 'meaning' of the link from the local resource (GML property element) to the remote resource; corresponds to the RDF notion of a property (*starting-resource HAS arc-role ending-resource*)
- **show**: used to communicate the desired 'presentation' of the remote resource on traversal from the starting resource; may take one of the values 'new', 'replace', 'embed', 'other', 'none'

6.2.2 Use of xlink in CSML

The pattern for xlink use in CSML exploits several of the available xlink attributes, and is illustrated in Figure 50.

```
<someGMLElement
  xlink:arcrole="hasRemoteContentEmbeddedAt#localXpath"
  xlink:href="storageDescriptor#portion"
  xlink:role="storageSchemaIdentifier"
  xlink:show="embed"/>
```

Figure 50: xlink pattern used in CSML

The *href* attribute points to a portion of a storage artefact – it could take any of several forms[6], e.g.:

- netCDFFile#variable: for identifying an individual variable within a netCDF file
- RDBMS#SQLQuery: for performing a SQL query against a relational source
- GRIBFile#recordNum: for identifying a particular record in a GRIB file
- CSMLStorageDescriptor#arrayID: for identifying a particular ArrayDescriptor instance within a CSML StorageDescriptor document

The *role* specifies the nature of the storage resource, .g. it could provide a mime type, or be a well-known URI identifying a particular resource type. A CSML StorageDescriptor will be identified through the URI "http://ndg.nerc.ac.uk/fileFormat/csmlStorageDescriptor".

The *arcrole* attribute specifies the 'nature' of the storage artefact relative to the CSML element from which the xlink originates. CSML exploits the syntax of URI (and bends the semantics!) to assert that the logical content described by the storage descriptor should be logically inserted into the CSML document at a point given by a particular relative XPath.

The mechanism is best illustrated by example. Consider a GML ReferenceableGrid instance. A portion might look as follows:

```
<csl:gridOrdinate>
  <csml:GridOrdinateDescription>
    <csml:coordAxisLabel>Geodetic longitude</csml:coordAxisLabel>
      <csml:coordAxisValues>
        <csml:SpatialOrTemporalPositionList>
          <csml:coordinateList srsName="WGS84">13.5 24.9 32.4 37.7 41.5 46.8 54.4 65.7</csml:coordinateList>
        </csml:SpatialOrTemporalPositionList>
      </csml:coordAxisValues>
    <csml:gridAxesSpanned>x</csml:gridAxesSpanned >
    <csml:sequenceRule axisOrder="+1">Linear</csml:sequenceRule>
  </csml:GridOrdinateDescription>
</csml:gridOrdinate>
```

Suppose however, that the longitude values in the coordinateList are stored in the 'lon' variable of a netCDF file, myfile.nc. Then the following application of xlink asserts this in CSML:

[6] Note that this is not conformant with the semantics of the URI specification, which requires the 'fragment' portion after the '#' separator to be evaluated only after the entire resource has been retrieved. In this case it provides merely a convenient syntax, understood to be targeted at some application near the data that can interpret the fragment portion.

```
<csml:gridOrdinate>
  <csml:GridOrdinateDescription>
    <csml:coordAxisLabel>Geodetic longitude</csml:coordAxisLabel>
      <csml:coordAxisValues
xlink:arcrole="http://ndg.nerc.ac.uk/xlinkUsage/insert#SpatialOrTemporalPositionList/coordinateList"
xlink:href="file://myfile.nc#lon"
xlink:role="http://ndg.nerc.ac.uk/fileFormat/netcdf"
xlink:show="embed">
        <csml:SpatialOrTemporalPositionList>
          <csml:coordinateList srsName="WGS84"/>
        </csml:SpatialOrTemporalPositionList>
      </csml:coordAxisValues>
    <csml:gridAxesSpanned>x</csml:gridAxesSpanned >
    <csml:sequenceRule axisOrder="+1">Linear</csml:sequenceRule>
  </csml:GridOrdinateDescription>
</csml:gridOrdinate>
```

The *arcrole* asserts that the netCDF 'lon' variable should be inserted at the relative XPath location 'SpatialOrTemporalPositionList/coordinateList' relative to the coordAxisValues start element. Such an XPath mechanism is needed since not all GML properties allow the 'by-reference' pattern (csml:coordinateList here is one example).

The *show* attribute, with its value of 'embed', means that the file-based content should logically be embedded within the CSML instance document.

7. TOOLING

Scanner and Parser are at http://proj.badc.rl.ac.uk/ndg/browser/TI02-CSML/trunk/csml

Documentation is at
`http://proj.badc.rl.ac.uk/ndg/wiki/UsingTheParserToCreateCSMLV2`

and `http://glue.badc.rl.ac.uk/ndg/wiki/CSMLParserHowTo`

8. SCHEMAS

Available at http://glue.badc.rl.ac.uk/ndg/browser/TI02-CSML/trunk/csml/XMLSchemas/

UML models at http://glue.badc.rl.ac.uk/ndg/browser/TI02-CSML/trunk/csml/UML

9. EXAMPLES

An example instance is at http://proj.badc.rl.ac.uk/ndg/browser/TI02-CSML/trunk/csml/XMLInstances/CSMLExample_SuperWrap.xml

10. APPENDIX: GML CHANGE REQUEST FOR REFERENCEABLEGRID

10.1 ReferenceableGrid

A referenceable grid is associated with a transform between grid coordinates and coordinates in an external coordinate reference system. Unlike a rectified grid, this relationship cannot be characterised through an affine transformation. It is instead provided in a table, relating the grid points to coordinates in the external coordinate reference system. The grid lines in the external coordinate reference system need not be straight or orthogonal, and the grid cells may be of different shapes and sizes.

EXAMPLE 1 Figure 51 shows an example of a referenceable grid.

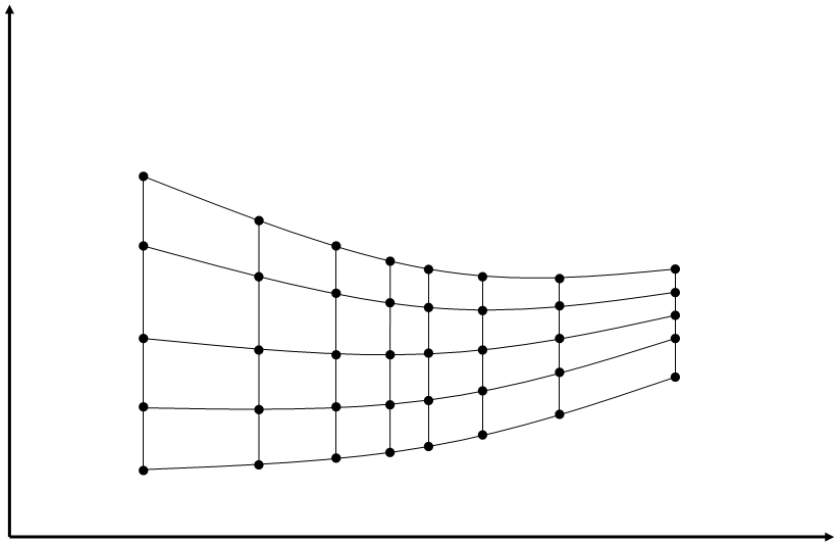


Figure 51: ReferenceableGrid example

Two mechanisms are provided to specify the transformation table – either listing the grid point locations explicitly as a sequence of direct positions in a defined sequence order (csml:GridPointDescription, 10.2), or listing the coordinate values for each coordinate system axis separately over the grid point locations (csml:GridOrdinateDescription, 10.3). The latter allows efficiency for the common case where an axis of the referenceable grid is aligned with a coordinate system axis.

The csml:ReferenceableGrid implements ISO 19123 CV_ReferenceableGrid (see D.2.11 and ISO 19123:2005, 8.10), and is defined as an extension to gml:Grid by adding a csml:coordTableTransform property of type csml:GridCoordsTableType:

```
    <element name="ReferenceableGrid" type="gml:ReferenceableGridType" substitutionGroup="gml:Grid"/>

    <complexType name="ReferenceableGridType">
        <annotation>
            <documentation>An implementation of CV_ReferenceableGrid of ISO 19123.
            The association role 'crs' to the SC_CRS to which it is referenceable is implemented using the
gml:SRSReferenceGroup attributeGroup inherited from gml:Grid.</documentation>
        </annotation>
        <complexContent>
            <extension base="gml:GridType">
                <sequence>
                    <element name="coordTransformTable" type="csml:GridCoordinatesTablePropertyType"/>
                </sequence>
            </extension>
```

```
        </complexContent>
    </complexType>
```

A property type for the csml:ReferenceableGrid is also defined:

```
    <complexType name="ReferenceableGridPropertyType">
        <sequence>
            <element ref="csml:ReferenceableGrid"/>
        </sequence>
        <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    </complexType>
```

The csml:GridCoordinatesTable is defined as follows:

```
    <complexType name="GridCoordinatesTablePropertyType">
        <sequence>
            <element ref="csml:GridCoordinatesTable"/>
        </sequence>
        <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    </complexType>

    <element name="GridCoordinatesTable" type="csml:GridCoordinatesTableType"
substitutionGroup="gml:AbstractObject"/>

    <complexType name="GridCoordinatesTableType">
        <annotation>
            <documentation>Defines a table providing the transformation between grid coordinates and external
coordinate reference system (see ISO 19123 8.2.1). Allows all grid point locations to be listed explicitly (using the
gml:gridPoints property), or individual ordinates to be listed for each coordinate system axis (using a set of
gml:gridOrdinate elements – one for each axis of the coordinate reference system).</documentation>
        </annotation>
        <choice>
            <element name="gridOrdinate" type="csml:GridOrdinateDescriptionPropertyType"
maxOccurs="unbounded"/>
            <element name="gridPoints" type="csml:GridPointDescriptionPropertyType"/>
        </choice>
    </complexType>
```

10.2 GridPointDescription, GridPointDescriptionType, GridPointDescriptionPropertyType

A csml:GridPointDescription lists the location of grid points explicitly as a sequence of direct positions in an external coordinate reference system, in a defined sequence order.

csml:GridPointDescription is defined as follows:

```
    <element name="GridPointDescription" type="csml:GridPointDescriptionType"
substitutionGroup="gml:AbstractObject"/>

    <complexType name="GridPointDescriptionType">
        <annotation>
            <documentation>GridPointDescription defines grid point locations over entire Grid by listing direct positions in
given sequence order.</documentation>
        </annotation>
        <sequence>
            <group ref="gml:geometricPositionListGroup"/>
            <element name="sequenceRule" type="gml:SequenceRuleType"/>
        </sequence>
    </complexType>

    <complexType name="GridPointDescriptionPropertyType">
        <sequence>
            <element ref="gml:GridPointDescription"/>
        </sequence>
        <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    </complexType>
```

The element group gml:geometricPositionListGroup contributes a gml:posList to the content model, providing a list of direct positions for all grid point locations, in the sequence order specified by gml:sequenceRule.

EXAMPLE 2 The csml:ReferenceableGrid of EXAMPLE 1 may be encoded using a csml:GridPointDescription as follows:

```
<csml:ReferenceableGrid gml:id="ID001" srsName="urn:ogc:def:crs:EPSG:6.6:4326" dimension="2">
    <gml:limits>
        <gml:GridEnvelope>
            <gml:low>0 0</gml:low>
            <gml:high>7 4</gml:high>
        </gml:GridEnvelope>
    </gml:limits>
    <gml:axisLabels>x y</gml:axisLabels>
    <csml:coordTransformTable>
        <csml:GridCoordinatesTable>
            <csml:gridPoints>
                <csml:GridPointDescription>
                    <gml:posList>
13.5 53.1  24.9 48.7  32.4 46.2  37.7 44.7  41.5 43.9  46.8 43.3  54.4 43.1  65.7 44.0
13.5 46.2  24.9 43.2  32.4 41.5  37.7 40.6  41.5 40.2  46.8 40.0  54.4 40.3  65.7 41.7
13.5 37.1  24.9 36.1  32.4 35.6  37.7 35.5  41.5 35.7  46.8 36.0  54.4 37.1  65.7 39.5
13.5 30.4  24.9 30.2  32.4 30.4  37.7 30.7  41.5 31.1  46.8 32.0  54.4 33.8  65.7 37.2
13.5 24.3  24.9 24.8  32.4 25.3  37.7 26.0  41.5 26.6  46.8 27.7  54.4 29.7  65.7 33.4</gml:posList>
                    <gml:sequenceRule axisOrder="+1 -2">Linear</gml:sequenceRule>
                </csml:GridPointDescription>
            </csml:gridPoints>
        </csml:GridCoordinatesTable>
    </csml:coordTransformTable>
</csml:ReferenceableGrid>
```

Figure 52: CSML ReferenceableGrid using GridPointDescription for geolocation

NOTE In this example the formatting of the grid point locations within the gml:posList is for illustrative purposes only and not relevant for XML processing.

10.3 GridOrdinateDescription, GridOrdinateDescriptionType, GridOrdinateDescriptionPropertyType

A csml:GridOrdinateDescription lists the values of a single coordinate system axis over the grid points in a defined sequence order; it is repeated for each axis of the coordinate system.

Furthermore, if these coordinate values are uniform along one or more axes of the grid, they do not need to be repeated over those grid axes. Thus, for the common case where a grid is aligned with the coordinate system axes (but has non-equally spaced grid point locations) the coordinate values of grid points need only be specified along relevant grid axes, and not over all grid points.

csml:GridOrdinateDescription is defined as follows:

```
<element name="GridOrdinateDescription" type="csml:GridOrdinateDescriptionType"
substitutionGroup="gml:AbstractObject"/>

<complexType name="GridOrdinateDescriptionType">
    <annotation>
        <documentation>GridOrdinateDescription defines grid point locations over entire Grid for a single spatial or
temporal coordinate axis.</documentation>
    </annotation>
    <sequence>
        <element name="coordAxisLabel" type="NCName"/>
        <element name="coordAxisValues" type="csml:SpatialOrTemporalPositionListPropertyType"/>
        <element name="gridAxesSpanned" type="gml:NCNameList"/>
```

```
            <element name="sequenceRule" type="gml:SequenceRuleType"/>
        </sequence>
    </complexType>

    <complexType name="GridOrdinateDescriptionPropertyType">
        <sequence>
            <element ref="csml:GridOrdinateDescription"/>
        </sequence>
        <attributeGroup ref="gml:OwnershipAttributeGroup"/>
    </complexType>
```

The element csml:coordAxisLabel names the coordinate system axis for which grid point locations are being defined. It should be consistent with the axis labels used for the CRS associated through the gml:srsName attribute of the parent csml:ReferenceableGrid.

csml:coordAxisValues lists grid point locations for the relevant coordinate system axis. Both spatial and temporal coordinate system axes are supported.

The element csml:gridAxesSpanned lists those grid axes along which this coordinate varies, and for which grid point locations are being prescribed. It will not normally include any grid axes along which all grid points have the same value for the coordinate being defined.

EXAMPLE 3 Consider a two-dimensional grid with axes x, y aligned with the longitude and latitude axes of the associated coordinate system. Then the longitude coordinate varies only along the x grid axis, and the latitude coordinate varies only along the y grid axis.

The element csml:sequenceRule specifies the order in which the coordinate values are being specified over the grid axes spanned.

EXAMPLE 4 The csml:ReferenceableGrid of EXAMPLE 1 may be encoded using csml:GridOrdinateDescription as follows:

```
<gml:ReferenceableGrid gml:id="ID001" srsName="urn:ogc:def:crs:EPSG:6.6:4326" dimension="2">
    <gml:limits>
        <gml:GridEnvelope>
            <gml:low>0 0</gml:low>
            <gml:high>7 4</gml:high>
        </gml:GridEnvelope>
    </gml:limits>
    <gml:axisLabels>x y</gml:axisLabels>
    <csml:coordTransformTable>
        <csml:GridCoordinatesTable>
            <csml:gridOrdinate>
                <csml:GridOrdinateDescription>
                    <csml:coordAxisLabel>Geodetic longitude</csml:coordAxisLabel>
                    <csml:coordAxisValues>
                        <csml:SpatialOrTemporalPositionList>
                            <csml:coordinateList>13.5 24.9 32.4 37.7 41.5 46.8 54.4
65.7</gml:coordinateList>
                        </csml:SpatialOrTemporalPositionList>
                    </sml:coordAxisValues>
                    <csml:gridAxesSpanned>x</csml:gridAxesSpanned >
                    <csml:sequenceRule axisOrder="+1">Linear</csml:sequenceRule>
                </csml:GridOrdinateDescription>
            </csml:gridOrdinate>
            <csml:gridOrdinate>
                <csml:GridOrdinateDescription>
                    <csml:coordAxisLabel>Geodetic latitude</csml:coordAxisLabel>
                    <csml:coordAxisValues>
                        <csml:SpatialOrTemporalPositionList>
                            <csml:coordinateList>
                                53.1 48.7 46.2 44.7 43.9 43.3 43.1 44.0
                                46.2 43.2 41.5 40.6 40.2 40.0 40.3 41.7
                                37.1 36.1 35.6 35.5 35.7 36.0 37.1 39.5
                                30.4 30.2 30.4 30.7 31.1 32.0 33.8 37.2
```

```
                        24.3 24.8 25.3 26.0 26.6 27.7 29.7 33.4
                    </csml:coordinateList>
                </csml:SpatialOrTemporalPositionList>
            </csml:coordAxisValues>
            <csml:gridAxesSpanned>x y</csml:gridAxesSpanned >
            <csml:sequenceRule axisOrder="+1 -2">Linear</csml:sequenceRule>
        </csml:GridOrdinateDescription>
    </csml:gridOrdinate>
</csml:GridCoordinatesTable>
</csml:coordTransformTable>
</csml:ReferenceableGrid>
```

Figure 53: CSML ReferenceableGrid using GridOrdinateDescription for geolocation

NOTE In this example the formatting of the coordinate values within the csml:coordinateList is not relevant for XML processing.

NOTE In this example, longitude coordinates of grid points are uniform along the 'y' axis of the grid (see Figure 8), and so the corresponding csml:gridOrdinate needs only to specify values along the 'x' axis (i.e. the grid axis spanned is 'x'). Latitude values, however, vary along both grid axes and so are specified over both grid axes in the defined sequence order.

In order to support compound spatiotemporal coordinate reference systems, coordinate values are specified using a cs:SpatialOrTemporalPositionList:

```
    <element name="SpatialOrTemporalPositionList" type="csml:SpatialOrTemporalPositionListType"
substitutionGroup="gml:AbstractObject"/>

    <complexType name="SpatialOrTemporalPositionListType">
        <annotation>
            <documentation>SpatialOrTemporalPositionList allows efficient lists of either spatial or temporal
positions.</documentation>
        </annotation>
        <choice>
            <element name="coordinateList">
                <simpleType>
                    <list itemType="double"/>
                </simpleType>
            </element>
            <element ref="csml:timePositionList"/>
        </choice>
    </complexType>
```

NOTE Spatiotemporal compound coordinate reference systems are explicitly intended to be supported for coverages over spatiotemporal domains (see the NOTE at GML clause 20.3.4). While the definitions above support referenceable grid geometries over such domains, additional changes are required to support rectified grid and other geometries over spatiotemporal domains (for instance gml:offsetVector does not support temporal offsets).

REFERENCES

[i] Woolf, A. *et. al.* (2003), "Data Virtualisation in the NERC DataGrid", UK e-Science All-Hands Meeting, Nottingham, UK [Online http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/094.pdf, 26 July, 2004]

[ii] A. Woolf, B. Lawrence, R. Lowry, K. Kleese van Dam, R. Cramer, M. Gutierrez, S. Kondapalli, S. Latham, D. Lowe, K. O'Neill, A. Stephens (2006), "Data integration with the Climate Science Modelling Language", Advances in Geosciences, **8**, 83 – 90. [Online: http://direct.sref.org/1680-7359/adgeo/2006-8-83, 31 January, 2007]

[iii] Woolf, A., Lawrence, B., Lowry, R., Kleese van Dam, K., Cramer, R., Gutierrez, M., Kondapalli, S., Latham, S., and O'Neill, K. (2005), "Standards-based data interoperability for the climate sciences", Met. Apps., **12**(1), 9–22, doi:10.1017/S1350482705001556.

[iv] ISO 19101, "Geographic information – Reference model"

[v] Andrew Woolf, Bryan Lawrence, Jeremy Tandy, Keiran Millard, Dominc Lowe, Sam Pepler (2006), "'Feature types' as an integration bridge in the climate sciences", AGU Fall Meeting, San Diego, Dec 2006, Eos Trans. AGU, **87**(52), Fall Meet. Suppl., Abstract IN53C-02.

[vi] ISO 19109, "Geographic information – Rules for application schema".

[vii] ISO 19110, "Geographic information – Methodology for feature cataloguing".

[viii] ISO 19136, "Geographic information – Geography Markup Language". [OGC GML version 3.1.1, document number 03-105r1, online: http://portal.opengeospatial.org/files/?artifact_id=4700, 31 January, 2007].

[ix] Atkinson, R. (2004), "Next steps to interoperability – Mechanisms for semantic interoperability", EOGEO Workshop, University College London, 23–25 June 2004, [Online: http://y.eogeo.org/files/eogeo2004/eogeo-2004-full-atkinson.ppt, 31 January, 2007]

[x] ISO 19107, "Geographic information – Spatial schema".

[xi] ISO 19111, "Geographic information – Spatial referencing by coordinates".

[xii] ISO 19108, "Geographic information – Temporal schema".

[xiii] ISO 19123, "Geographic information – Schema for coverage geometry and functions".

[xiv] ISO 19103, "Geographic information – Conceptual schema language".

[xv] ISO 19118, "Geographic information – Encoding".

[xvi] UK MetOffice (2006), "OGC Web Services and GML Modelling for Operational Meteorology", Communiqué – workshop findings and outcomes [Online: http://www.metoffice.gov.uk/informatics/OGC_Web_Services_and_GML_Modelling_for_Operational_Meteorology--communique_V1.pdf, 31 January 2007]

[xvii] Cox, S., *ed.* (2003), "Observations and Measurements", OpenGIS Consortium document 05-08r4 [Online: http://portal.opengeospatial.org/files/?artifact_id=17038, 31 January, 2007].

[xviii] Woolf, A., "Climate Science Modelling Language Version 0.1 User's Manual", 9 Jan. 2005 [Online: http://ndg.nerc.ac.uk/csml/UsersManual.pdf, 31 January 2007]

[xix] Andrew Woolf, Bryan Lawrence, Row Lowry, Kerstin Kleese van Dam, Ray Cramer, Marta Gutierrez, Siva Kondapalli, Sue Latham, Kevin O'Neill, Ag Stephens (2005), "Climate Science Modelling Language: standards-based markup for metocean data", 85th Annual Meeting of the American Meteorological Society, San Diego (Jan 2005).

[xx] Object Modelling Group, "UMG Unified Modeling Language Specification", Version 1.5, Mar. 2003.

[xxi] Woolf, A., "GML Change Request for ReferenceableGrid", OGC document 06-160r1, 20 Nov. 2006 [Online: http://portal.opengeospatial.org/files/?artifact_id=18806&version=1, 1 Feb 2007]

[xxii] W3C (2001), "XML Linking Language (XLink) Version 1.0", 27 June 2001 [Online: http://www.w3.org/TR/xlink/, 2 Feb. 2007]]